Report: T2000:14
ISRN: SICS-T–2000/14-SE
ISSN: 1100-3154

TCP Performance in Ad Hoc Networks

by

Mattias Östergren

November 15, 2000

mattiaso@sics.se

Swedish Institute of Computer Science
Box 1263, S-164 29 KISTA, SWEDEN

Abstract

Ad hoc networks are mobile wireless networks which do not have any kind of fixed infrastructure. The routing layer in an ad hoc network ties the network together into a seamless entity and provide transparent services to higher layer protocols. This thesis examines the interactions of two routing protocols, AODV and DSR and how the mobile ad hoc network environment affect TCP performance. The results presented here are as follows: the path length and the presence of competing traffic are the main factors of TCP throughput performance. The size of TCP window affects the loss rate, but the loss rate is not strongly correlated to throughput performance. Using TCP selective acknowledgement option does not improve throughput. Finally, there is hardly any difference in TCP throughput when using DSR and AODV. These conclusions are supported by extensive simulation experiments.

Keywords: Ad hoc Networks, AODV, DSR, TCP Performance

Contents

1 Introduction

An ad hoc network is a mobile wireless network operating in an environment where there is no fixed infrastructure. Each node is equipped with a radio device which permits it to communicate with other nodes within range. However, a node outside this range is unreachable unless it is in indirect contact with a node within range. Communication is possible along chains of nodes, where each adjacent pair of nodes are in direct contact. Along this chain data is forwarded in a multi-hop fashion. All nodes carry the ability to route and forward traffic through the network. Ad hoc networks have the potential of establishing connectivity spontaneously, without any manual configuration effort at the routing layer. Also, the routing layer in an ad hoc network has the ability to reconfigure since nodes may move about, causing changes to the topology [2, 6, 11].

An example of a situation where the use of an ad hoc network is interesting is conferences, where participants need to exchange information stored in their personal digital assistants and there is no time for lengthy configuration procedures. Another situation is search and rescue operations where orders and information effectively is distributed among crew members.

A fairly well-studied aspect of ad hoc networks are routing protocols and their relative performance [7, 8, 9]. Little is however done in evaluating how well the different routing protocols interact with transport level protocols, such as the Transmission Control Protocol (TCP) [14]. The only study that measures the performance of TCP in an ad hoc network is made by G. Holland and N. Vaidya [21]. Their study is different from this thesis in that they tested TCP performing a single transfer. They chose a simulation environment where there was no competing traffic in the network and the measurements were made in a network routed with only one ad hoc routing protocol, the Dynamic Source Routing (DSR) protocol [12].

This thesis provides a broad study of TCP performance in Ad Hoc Networks. TCP performance is tested in networks routed with two routing protocols, DSR [12] and Ad hoc On Demand Distance Vector (AODV) [13]. Furthermore, two version of TCP, TCP/Reno and TCP/Sack, are tested under wide set of parameters performing multiple concurrent transfers. In order to evaluate the effect of the parameters, three metrics are calculated. They are throughput, loss rate and router queue length.

The rest of the thesis is outlined as follows. Section 2 gives an introduction to the Ad Hoc Network environment. Section 3 and 4 extend this introduction by describing a selection of Medium ACcess (MAC) layer and ad hoc routing protocols. Section 5 gives a brief description of TCP and various TCP mechanisms and options. Section 6 concludes the introduction by

giving an overview of the simulator environment used in the experiments. In Section 7 the chosen TCP performance metrics are defined and discussed. Experiments are presented in Section 8. In Section 9 the thesis is ended with a conclusion.

2 Ad Hoc Networks

The term Mobile Ad Hoc NEtworking (MANET) captures the concept of an autonomous wireless mobile network, where the network has no fixed infrastructure [2]. Nodes communicate via a broadcast mechanism if the destination is within range and forwards data to other destinations in a multi-hop fashion. The ability to route traffic is distributed to all nodes in the network. A MANET has the potential of being able to form networks spontaneously, without active configuration effort at the routing layer. Furthermore, the routing layer has the ability to handle the frequent network topology reconfigurations due to node mobility. Finally, a MANET may be heterogenous since the nodes may offer services of varying characteristics.

Being autonomous, a MANET requires little or no configuration and maintenance in order for its nodes to work together. The nodes connect to each other by themselves, forming the network *ad hoc*. Since a MANET is wireless, each node posess a device that is being able to transmit and receive electromagnetic waves. However, some nodes may have several other communication devices, which may be connected to a fixed network to provide Internet access. A MANET is mobile, which means that nodes in the network are free to spatially move around, completely arbitrarily. At every instance of time, depending on the position of each node and the range of each transmitter device, a connectivity pattern in the shape of a graph is formed. The routing layer at each node makes sure that they can communicate to any other member of the same graph, while moving. A MANET does not need any kind of fixed infrastructure. There is no central administration such as a base station or router hardware. All the functionality of the network is completely distributed to each node in the network.

A MANET has several other characteristics. The most prominent ones are the limited availability of bandwidth, energy, and security [2]. Due to the variable capacity of the devices and that wireless transmission techniques are sensitive to noise and interference, which introduces transmission errors, the available bandwidth may be considerably lower than in a wired counterpart. The multiple access techniques used at the physical layer, in order for the nodes to share the medium, lowers the capacity further. Some nodes in a MANET may run on a limited energy supply. If a node is driven by a battery

it should not take part in all the communication going on in the network or it will soon cease to operate. Exchanging messages over a wireless link is inherently more insecure than doing it on a wired link. The wireless link is easy to eavesdrop, spoof or otherwise disturb by denial-of-service attacks.

3 Ad Hoc Medium Access Control

Due to the salient characteristics of a MANET, a specialized Medium Access Control (MAC) layer must be used when transmitting data. Being wireless the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) [4, page 252] technique, used extensively in wired networks, is not applicable. This technique works on the assumption that if a collision occurs while sending a packet, the collision will be detected by the node that transmitted it. It is the job of the transmitter to notify other nodes of a collision by sending a jamming signal and retransmit the lost packet. This assumption is not valid in a wireless network when the situation called hidden terminal [3, page 62] is in effect.

The hidden terminal effect happens when there are three nodes, A, B and C. In this scenario A and B and nodes B and C are within wireless range of each other, but nodes A and C are not. The CSMA/CD technique breaks down when A sends to B and C also wants to send to B. C will sense the medium and since A is out of range it will conclude the medium is free and start sending. The data is garbled at B. However, the signals from C does not reach A, and node A will never be aware that data is lost at B. Node A will never realize it must retransmit the lost data.

A MAC layer scheme called Multiple Access with Collision Avoidance (MACA) [26] solves the hidden station problem without using fixed infrastructure. It is thus well suited for the MANET environment. With MACA, a data transfer has to be preceded by a messages exchange. Given the same situation as above, when node A wants to send to node B, node A first transmits a Request To Send (RTS). If B is prepared to receive data, it grants A access to the medium by sending a Clear To Send (CTS). If any other node waiting to send data to B hears either the RTS or CTS, it must remain silent until B is finished.

3.1 IEEE 802.11

The IEEE 802.11 specifies a Wireless LAN [5] standard. In this standard there is a protocol based on the MACA principle suited for a MANET. The protocol divides time into “frames”. Stations can contend for the medium

only during a part of this frame. In between two consecutive frames there is a mandatory idle time. The parameter Distributed coordination function Inter-Frame Spacing (DIFS), governs the default idle time or the lowest priority to the medium. The Short Inter-Frame Spacing (SIFS) is the shortest idle time and signals high priority to the medium.

Assume that we have two stations A and B, and that station A wants to unicast to station B. If the medium is busy when station A wants to send it has to wait for DIFS and enter the contention phase. Station A picks a random back-off time within a dedicated contention period. A station wins the contention when the back-off time ends and the medium is not busy. This station is now free to transmitt data. If, however, another station wins the contention, station A saves the remaining amount of back-off time. In the next contention period, station A waits this amount of time rather than picking a new random back-off time.

When A wins the contention it transmits a RTS message. The RTS message contains the address to B and the amount of time that A has to spend in order to transfer a packet. Every node, other than B and A, has to set its Net Allocation Vector (NAV) to the earliest point in time they can contend for the medium in accordance the RTS message. B waits until the SIFS phase is over and sends the CTS. The CTS indicates that B is ready to accept a data transfer from A and how long this transfer will proceed. All other stations, except A and the stations disabled due to NAV_{RTS} or out of wireless reach, that hear the CTS, immediately update their NAV. Since there is no risk of collision, A is free to send the data to B. Upon reception of the data and after waiting for SIFS, B sends an acknowledgement to A.

IEEE 802.11 reports a send failure to the higher layer either when it does not get a CTS in response to several RTS, or if it does not receive an acknowledgement on a data packet. The send failure indication is used extensively in some ad hoc routing protocols.

If a node wants to broadcast a message, the node first sense the medium. If the medium is free, the node broadcasts the message with no further notice. A node overhearing the broadcast does not send back an acknowledgement.

4 Ad Hoc Routing Protocols

The purpose of a routing protocol is to provide means of communication between computers on different but inter-connected networks. In a wired network, it is the router that provides this functionality. The router is essentially a device that has one connection to the local network and one or more connections to other remote networks. When it receives a packet, tagged with

a remote address, it usually consults a routing table. The table maps a remote address to the next hop along the path. The forwarding ability of a router, usually consists of a routing table look-up. In order to correctly forward packets at all times, a router maintains its table by running a routing protocol. The process in which a routing protocol updates the mappings of the router table is called routing.

In MANETs routing and forwarding schemes are also needed. The source of a data transfer and the sink may not be in direct wireless range of each other, but a path, traversing the connectivity graph, in one ore more wireless hops, may still exist. The path can be used to establish an end-to-end connection. However, a MANET is different from a wired network. Since there is no fixed infrastructure, in order to connect a source and sink on a path, each node in the MANET must have the same functionality as a router. Thus each node is essentially a combination of a router and a host. When a node either originates or receives a packet stamped with a destination other than itself, it forwards the packet to the next node on the path. In order to maintain correct a view of the shape of the connectivity graph, each node does routing.

Here, two ad hoc routing protocols, Ad Hoc On demand Distance Vector routing (AODV) and Dynamic Source Routing (DSR), will be investigated. Using the taxonomy introduced by L. M. Feeney [11] AODV is a single channel, uniform, destination based, reactive routing protocol. DSR, on the other hand, is a single channel, uniform, topology based, reactive routing protocol.

A single channel routing protocol works on the assumption that the link layer communicates over a single logical channel. The design of such a channel is often based on the MACA principle. If a routing protocol is uniform, there is no hierarchical structure in the routing procedure. All nodes have the exact same ability to route packets, that is, the routing procedure is fully distributed. A topology based protocol tries to maintain large scale topology information in order to make routing decisions. The most widely known topology based routing strategy is to make each node periodically advertise their link state information. Then the shortest path to each destination is computed and stored in a route table. A destination based protocol does not keep large scale topology information. The most widely known class of destination based routing are the distance-vector protocols. The routing information kept at each node maps each known destination with a distance in hops and a vector telling the next hop. If a routing protocol is reactive the route to a particular destination is obtained only if it is needed at the time when it is needed. The routing schema of reactive protocols builds on two processes, route discovery and route maintenance.

The route discovery is executed when a node needs a route to a new destination. The network is then flooded with a special Route REQuest message

(RREQ). Nodes use various schemes to limit the flood rate of broadcasts. At each node the RREQ visits, all the routing information back to the originator is stored. When the route request reaches the destination a Route REply (RREP) is constructed and unicast back to the originator of the route request. As the RREP works its way back to the source on one path of nodes visited by the RREQ, routing information to the destination is stored. Upon the receipt of the RREP, the source can start to forward data. Route maintenance sees to that stale routes are removed, possibly re-initiating the route discovery.

4.1 AODV: Ad Hoc On Demand Distance Vector Routing

AODV [13] is a routing protocol in a strict sense - there is no byte overhead when forwarding. It based on the widely known distance vector algorithm. The distance is measured in wireless hops and the vector is the next hop node on the path to the destination. In order to better adopt the distance vector algorithm to topology changes, each node keeps a Destination Sequence Number (DSN) enforcing a order on each route using this node as destination. The higher the DSN, the more recent and thus the more valid the route to it is. To further improve AODVs performance, a route is only considered valid for a limited period of time. Each time a node uses a route its lifetime is updated. If the lifetime of a route expires, route discovery may be initiated or, if no need for the route is present, it may be deleted. Finally, each route table entry carries a precursor list. A precursor list is a list of nodes which forwards packets on this route. The purpose of keeping a precursor list is that they will receive a route error notification on a loss of a next hop link.

4.1.1 Route Discovery

Being on-demand a node initiates route discovery when it needs a route to a destination. The reason for this need is because the node has not been forwarding data to the destination before, or a previously used route has become invalid. The route discovery process consist of generating a RREQ and then waiting for a RREP. All packets addressed to this destination have to be buffered until a route is present.

A node uses two techniques in order not to flood the network with unnecessary broadcasts. The first is to include a unique Broadcast ID (BID) in the RREQ. The second is to use an expanding ring search. The expanding ring search is implemented by querying all the nodes for a destination, within a ring with a radius set to a number of wireless hops. A node starts by

querying its immediate neighbors one hop away. If this test fails, the radius is increased and the procedure is repeated until the node is found. If the node is not the destination and has no route for it it must rebroadcast the RREQ. The outgoing RREQ is set up to use this node's IP address in the IP header, allowing the next hop node to create a reverse route. Before broadcasting the RREQ the node creates or updates a reverse route to the originator of the RREQ. A node discards a RREQ received, if it has previously received a RREQ from the same originator with the same BID. For all other RREQs, a node checks if it is the destination and if it has an active route for the destination.

If the node is not the destination but has a route for it, the node compares the DSN in the RREQ with the DSN in the route cache. If the DSN in the RREQ is less than or equal the node generates a RREP on behalf of the destination. However, if DSN in the RREQ is greater, the RREQ is rebroadcasted. When generating a RREP on behalf of the destination, a node updates both the precursor list for the destination and the precursor list for the reverse route to the source. The node copies over the DSN found in the route entry to the RREP.

When a node receives a RREQ and it is the destination, it creates or updates a reverse route towards the originator. The destination puts a unique DSN in the RREP. A unique DSN is generated by incrementing a variable by at least one on each local connectivity change.

When a node receives a RREP it creates a forward route to the destination. If the node is not the originator of the corresponding RREQ, the node forwards the RREP, using this nodes IP address in the IP header, on the reverse route.

4.1.2 Route Maintenance

On a connectivity change AODV's route maintenance procedure is started. It consists of generating Route ERRor (RERR) messages and possibly restarting the route discovery. AODV may be configured to use link level notifications of connectivity changes. When a change happens for an active next hop the node checks the precursor list for it. If the precursor list is non-empty, it calculates which destinations are unreachable due to the loss, and broadcasts a RERR. When a node receives RERR, it checks if it is on a path to any unreachable destination. If so, it marks this destination as unreachable. For each destination it checks its precursor list and includes every destination with a non-empty precursor list in a new RERR.

4.2 DSR: Dynamic Source Routing

DSR [12] uses a forwarding technique called source routing, which means that all the routing information necessary to reach the destination is provided by the source. The route information consist conceptually of a list of addresses to nodes which must be visited in order to reach the destination. This list is enclosed in each packet forwarded.

4.2.1 Route Discovery

Since DSR is a reactive routing protocol, route discovery is initiated only if there is a demand for a route, that is, there has to be at least one packet waiting in the send buffer addressed to an unknown destination to which a route does not exist. The route discovery process consists of sending RREQ and waiting for a RREP. Several route aquisitions may be serviced in paralell. When a RREP comes in, the node sends all packets waiting for a particular route. The first step in the route discovery is to generate a non-propagating RREQ. It is distributed to all nodes within the radius of one wireless hop. The purpose of a non-propagating RREQ, is to prevent the RREQ to spread network wide, wasting bandwidth on unnecessary broadcasts. It is quite possible that the immediate neighbors already have a route to the destination in their caches. If no RREP is received in response to the non-propagating RREQ, a propagating RREQ is sent. A propagating RREQ is allowed to be distributed network wide. The process of sending propagating RREQs is repeated until the destination is found.

When a node receives a RREQ it must first check if it is the target of the route discovery. If so, it takes the route found in the incoming RREQ and copies it over to a RREP. If the destination has a route to the originator of the RREQ, it unicasts the RREP jitter-ed by a random number of time. If the route is not present, the destination performs a route discovery piggybacking the RREP. The piggybacking prevents loops, where the originator of the request has to perform yet another route discovery to answer this route discovery.

Otherwise, if the node is not the target of the route discovery, it checks if it has received a RREQ from the same originator with the same Request IDentity (RID). If this is a duplicate the node silently discards it. If this is the first RREQ received the node checks if there is a route present in its cache. If not, the node appends its address in the list of addresses and the RREQ is broadcasted.

If there is a route in the cache, the node is allowed to send a RREP on behalf of the destination. The node constructs a complete route from the

originator to the destination by appending the route in the route cache with the route in the RREQ. If the complete route does not contain any loops and has this node as a next hop somewhere along it, the node sends the RREP.

Each node, that processes the RREP, extracts all the route information possible and incorporates into its cache. The node then checks if this information can release any packets from the send buffer.

4.2.2 Route Maintenance

Whenever transmitting a data packet, a node has to make sure it is received by the next hop. DSR can be configured to use link layer notification and thus assumes all transfers are correct until a notification occurs. When a failure happens a Route ERRor message (RERR) is constructed and sent. The RERR is sent back to the originator of the packet, performing a route discovery first, if necessary. When a node receives a RERR it removes all routes that visits this node.

After generating a RERR an intermediate node may try to salvage a packet by checking the route cache. If a node does have an alternative route it constructs a new data header with this route, else it is dropped. Since wireless transmission is inherently broadcast, each node may configure its hardware not to filter packets. As an optimization a node may then update its cache with any routing information it overhears.

5 TCP: Transmission Control Protocol

The Ad Hoc routing protocols bring a network level consistent view of the MANET, but this view does not provide a reliable end-to-end transfer service. TCP [14] is a transport layer protocol responsible for reliable end-to-end data transportation. It is the most widely used reliable end-to-end transport protocol on the Internet. The use of TCP in MANET is necessary since it would enable a lot of popular applications and provide integration with the Internet.

TCP is a connection oriented, reliable byte stream protocol. Being connection oriented, TCP is usually configured to set up a connection before any data can be sent. The setup phase consist of running the connection establishment algorithm, which is an implementation of a three-way handshake. The three-way handshake is a process of exchanging a series of messages. First the originator sends a request to establish a connection. The destination acknowledges this request and the originator acknowledges the acknowledgement. When the connection is established data can flow in both

directions.

TCP divides the incoming byte stream into segments. The operation of the transfer is based on these segments. In order for the source to conclude that a particular segment has been received, it must get an acknowledgement from the sink. In order to prevent TCP from waiting for acknowledgement indefinitely, a timer is started when a packet is sent. If the timer expires the segment is retransmitted. Furthermore, to be able to detect corrupted segments, a check-sum is calculated and enclosed in each segment before it is sent. Upon reception, if the checksum is invalid, the segment is discarded.

In TCP both ends of the connection can close it independently of each other. One end of the connection may still have data left to send when the other is finished. The end that is finished may then terminate its connection to signal it is done sending data. The closing of a connection is handled by the connection termination algorithm. This procedure allows any side of the transfer to close its forward direction of data. The end that decides to close sends a message indicating this, the receiver acknowledges this message. However the originator of the connection close indication must still be around to acknowledge the termination of the other end. All in all, two close procedures are performed.

5.1 TCP flow control

There are several mechanisms and options that are used when a TCP transfer is performed in order to control the flow of data. The mandatory mechanisms are slow start and congestion avoidance [15]. Popular options are fast recovery, fast retransmit [15] and selective acknowledgements [16]. Due to historical reasons when slow start, congestion avoidance, fast recovery and fast retransmit are used in conjunction, the TCP version is called TCP/Reno [14, page 17]. A TCP version that uses the selective acknowledgement option in addition to slow start, congestion control, fast recovery and fast retransmit, is called TCP/Sack.

If TCP is too aggressive and sends too much data into the network, the router queues soon build up and start loosing packets. This phenomenon is called congestion. In order to limit the initial send rate, TCP must use slow start. To prevent congestion, TCP uses an algorithm called congestion avoidance. These two algorithms limit the amount of data the TCP source can send before getting an acknowledgement.

Slow start begins with a window size of one segment. For each acknowledgement, slow start increments the size of the window by one segment. This has the effect of increasing the amount outstanding data exponentially over time. Slow start is used until either the amount of outstanding data exceeds

a threshold value, congestion occurs or the maximum size of the window is reached. At this point, if there is no congestion and the maximum window is not reached, the congestion avoidance increments the amount of outstanding data by one segment for each round-trip. This has the effect that the amount of outstanding data is increased linearly over time. When the maximum window size is reached the amount of outstanding data is not increased further.

Whenever congestion occurs, as indicated by a time-out, the threshold value is set to roughly half the window size. The amount of outstanding data is set to one segment and then slow start is run until the window reaches the new value of the threshold. At this point congestion avoidance takes over.

Slow start and congestion avoidance are flow controlling mechanisms performed by the sender. The amount of outstanding data may also be limited by the receiver. If the receiver is running out of buffer space, it may advertise a reduction in the amount of data sent by the sender.

When TCP receives three duplicate acknowledgement packets, it retransmits what seems to be the missing segment without waiting for the retransmission timer to expire. This is called fast retransmit. After the missing segment has been transmitted the fast retransmit procedure steers the transmission of new data, until a non-duplicate acknowledgement is received. This procedure sets the threshold value to roughly half the window size. This value is saved for future reference. Then the window is set to the value of the threshold plus three segments. As long as the sender receives duplicate ackowledgements the window is incremented by one segment. When the first non-duplicate acknowledgement is received the threshold is restored to the saved value.

Selective acknowledgement is an extension to the acknowledgement scheme used in the TCP transfer. In TCP versions that does not use the selective acknowledgement option, the sender has only a coarse-grained knowledge of what segments have been received. It can only determine one missing packet to retransmit per round-trip. If multiple segments are lost, TCP faces a lot of extra work, possibly retransmitting a lot of packets already received. The idea with selective acknowledgement is to remove the unnecessary work the packet loss incurs. In this algorithm, the sender is informed which packets are missing at the receiver. The sender then retransmits only these missing packets. TCP selective acknowledgement uses an extended acknowledgement format. This format allows the receiver to notify the sender that up to six *holes* of missing segments exist in the segments queued at the receiver.

6 Monarch's Extension to ns-2

In order to measure TCP performance a simulator called Ns [17] is used. Ns is a discrete event network simulator. The original Berkeley implementation of it only simulates stationary wired networks - the CMU Monarch project's extension [18] adds functionality in order to simulate MANETs. The new functions are: node mobility; radio propagation models; and ad hoc routing protocols.

A scenario, a communication pattern and choice of routing protocol files are fed to the simulator which produces a trace file. The pattern, in which a node moves, is specified in the scenario file. Since all of the nodes are assumed to run the same ad hoc routing protocol, the routing protocol chosen is specified with a parameter to the simulator. The communication pattern specifies how the data transfers will be performed in a particular simulation.

6.1 The Architecture of the Simulator

The simulator is organized as a collection of independent nodes. Each node runs a network stack organized into four levels: the physical layer, the link layer (or MAC layer), the routing layer and the agent layer.

The physical layer simulates hardware and signals. Propagation of electromagnetic waves is simulated using a combination of free space propagation and a two ray ground reflection model [25]. In the free space model, a signal is assumed to propagate without interference from the sender to the receiver along the line of sight. The signal attenuates, that is the power at the receiving antenna decreases proportionally to $\frac{1}{r^2}$, where r is the distance to the sender. However, if the signal is transmitted over the earth surface, some rays will bounce and cancel others and lowering signal strength. In the two ray ground reflection model, one ray is assumed to bounce off the flat ground and cancel the other ray traveling along the line of sight, causing an attenuation to the signal proportional to $\frac{1}{r^4}$. The simulator combines the two models using the free space model at short distances and the two ray reflection model at longer distances. Packets may be lost or thrown away upon reception due to collisions or weak signals, but they are never modified. The simulator does not model any kind of transmission errors.

The link layer is internally split up in several sub parts. The most significant parts are: the MAC module, the interface queue, and the Address Resolution Protocol (ARP) [14, chapter 4] module. The MAC module is an implementation of IEEE 802.11 as described above. The interface queue is a drop-tail FIFO queue holding packets waiting to be transmitted.

The interface queue is divided into four sub-queues with different send

priority. The four sub queues share the capacity of holding a maximum of 50 packets, independent of size. The sum of the length of each sub queue may not exceed 50 packets. Router layer messages are stored in a sub-queue with the highest priority. TCP acknowledgements are stored in the sub queue with the intermediate send priority and TCP data packet are inserted in the queue with the lowest priority. The fourth queue is used by real-time traffic. The reason for having a multi-level priority of packets is to make sure that routing protocol control messages are delivered as quickly as possible, which is crucial for performance [10]. The TCP acknowledgements precede the data packets in send order to avoid that the sender's timers expire too soon. The routing layer has full access to the queue and is able to insert, delete packets, and otherwise manipulate the queue as it needs. This right is used extensively, especially on link failures.

The function of the ARP module is to act as a mapping of IP addresses to MAC layer addresses. ARP works by broadcasting a request for a MAC address for a given IP address. The node with the matching IP address then replies to this request. The protocol is initiated on-demand and mappings are stored in a cache to prevent unnecessary lookups. The ARP request/response exchange is limited to deal with only one packet at the time. If two packets are sent to the ARP module and both packets require lookup to the same destination, the packet arriving first will be replaced by the second.

The routing layer roughly corresponds to the network layer in the ISO stack. The detailed operation of this level is specific for each routing protocol. The implementation of each protocol is an adaption of the Internet drafts to the simulator environment. Currently the simulator implements Internet draft version 3 of DSR and version 4 of AODV.

At the transport layer we have TCP. In the simulator a TCP transfer is set up between two nodes, the source and the sink. The transfer is simplex, data is flowing in one direction and only acknowledgements in the opposite. There is no connection establishment or termination. Since there is no concept of user level processes, data is immediately dealt with on reception and need not to be stored until user process is ready to read. This means that there are no receiver window advertisements in the acknowledgements, going back to the source. In order to limit the window, the number of segments sent by the source is taken as minimum of the congestion window variable and the maximum allowed window size.

The agent layer in the simulator corresponds to the application layer in the ISO network stack. The agent is the originator of data packets in the simulator. The simulator implements many agents, but the experiments presented here are only concerned with one, which is called FTP. It mimics a simple file transfer protocol.

6.2 Simulator Input and Output

The scenario files specify how the nodes move about. Each node is associated with a position and a destination. Also, each node moves with a certain speed until it reaches its destination. The destination may change before the node actually reaches the first one, and a node is not required to ever reach a destination. If, however, a node actually reaches its destination, it pauses for a specified amount of time. Then, if there are any remaining movement instructions, the node heads for the next destination. This process is repeated until there are no more movement instructions for this node. When a node reaches the end of its movement pattern its stops moving and pauses until the end of the simulation.

The communication pattern file initiates the operation of the agent level at each node. This file specifies what kind of agent level agents are used and how they participate in the pattern. Also this file defines which nodes are involved in the communication.

The output of the simulator is called a trace file. It contains information derived from the different levels in the networks stack during simulation. It is crucial to understand what and when information is being logged in order to fully understand and validate experiments. A thorough explanation of the trace file format can be found in Appendix B.

6.3 Bugs in the Simulator

Three bugs were discovered in the software package. One bug was found in the code that simulates the interface queue in the MAC layer. The second bug was found in the implementation of DSR and the third in the implementation of AODV. The bugs are confirmed by the group developing the simulator and DSR at Carnegie-Mellon University and the group developing AODV at Cincinnati University. Fixes to these bugs are being incorporated in the corresponding software releases. It took approximately three weeks to find the bugs, report them and then recieve fixes or develop custom fixes. The bug that was hardest to find was contained in the MAC layer. The DSR bug was quite obvious while the AODV bug was considerable harder to detect, since it only occurs in rare cases.

The code implementing the interface queue contained a bug that prevented some packets to be extracted on a link failure. The code that traverses these queues to extract a packet starts on the sub queue with the lowest priority. If there are no packets in this queue going on this bad link, the code jumps to the next queue. This procedure is iterated for all sub queues until a packet is found. When a packet is found the inner search is stopped, but the

outer will force the search to continue in other queues. Assume that when the link failure happens there are only packets in one of the first three queues. The outer search selects a queue and the inner search then finds a packet to extract. The inner search is halted but the outer continues to the next queue. Due to implementation details, when the entire search procedure is finished, that is all remaining queues are examined, the information of which packet to extract is lost.

In DSR, a bug in the implementation causes all but the first and the last packets in the interface queue to be dropped. The problem occurs when there is a link failure. The link layer notifies that a packet could not be transmitted on a particular link. The code then fails to pick out all the other packets waiting in the interface queue, going on the same link. When the packets are salvaged, only the packet that was being sent when the link failed and the last packet of the packets waiting are reinserted.

In AODV the code to increment Destination Sequence Number (DSN) contained a bug that caused the transfer to halt for about ten seconds. The implementation broke down in the following case. A source node and a sink node engages in a TCP transfer. Assume that the DSN is one for both nodes and that the optimal hop count is one. The nodes are within reach of each other. Also, assume that nodes are not involved in any other kind of traffic that generates route discoveries, which would increase their DSN. Then, after the connection setup phase, which involves two route discoveries, the nodes involved in the transfer will have exchanged knowledge of the others DSN.

When the transfer is in process both nodes always have to forward data both directions. This means that when a link failure happens, both nodes will be notified of the problem, and both ends will take actions to recover from it almost at the same time. Assume that the link failure makes the path between the sender and receiver one hop longer. There is at least one intermediate node. Since the source has not been involved in any other route discoveries, it marks the route to the sink as being down and increments the DSN by one. The source constructs a route request RREQ_s and sends it.

Due to interleaving, the sink discovers the route in question is broken before it receives a RREQ_s from the source. It then marks the route as being down and increment the sequence number for this node. The sink then initiates route discovery. A route error message RERR_d and a route request message RREQ_d is generated. The highest known DSN for this route, which is two, is associated with RREQ_d.

The sink then gets RREQ_s, marked with DSN of two, before it has the reply to its own route request. It does not update the route entry properly. The DSN in RREQ_s is equal to the DSN in the route cache, but the hop count in RREQ_s is bigger than the hop count in the route cache, by at least

one. Due to a bug in the software, the route is not updated.

Then when the sink constructs a route reply, it makes a lookup in the route cache, to find the next hop node. It finds a route to the source, which is marked down and the next hop value set to the node that it cannot reach. Still it uses the information in the route entry to set up the reply. The route reply is then stuck in the send queue.

Before the reply can be sent, the sink broadcast the RERR_d followed by RREQ_d. The neighboring nodes answer the request, but the DSN in their replies are also two and the route cache is not updated. The operation would now halt until a internal timer that governs the rate of route discoveries would expire.

7 TCP Performance Metrics

Three metrics are calculated in this thesis in order to compare and evaluate performance of different TCP versions in various MANET settings. The metrics are throughput, loss rate and router queue length [19].

7.1 Throughput

The most obvious way to measure TCP performance is to measure the throughput, or the time required to send a given amount of bits. In TCP, a packet is considered successfully received when it is acknowledged at the sender. Therefore, throughput is defined to be the time to acknowledge a given amount of payload bits at the sender. In order to estimate the impact and effectiveness of the MANET environment in a given scenario, a routing protocol independent metric, the expected throughput [21], is defined and calculated. The values are then compared to the measured throughput.

7.1.1 Expected Throughput

Let t_i be the duration in time in which the shortest path between the sender and receiver is i hops. Let T_i be the the throughput for a certain set of TCP parameters obtained in a network with no routing overhead, when the shortest path is i hops. The amount of data sent at a certain hop length i is then $t_i T_i$. The *expected* throughput, is the sum of the amount of data sent at each hop length over the total transfer time [21]:

$$B_{exp} = \frac{\sum_{i=1}^{\infty} t_i T_i}{\sum_{i=1}^{\infty} t_i} \tag{1}$$

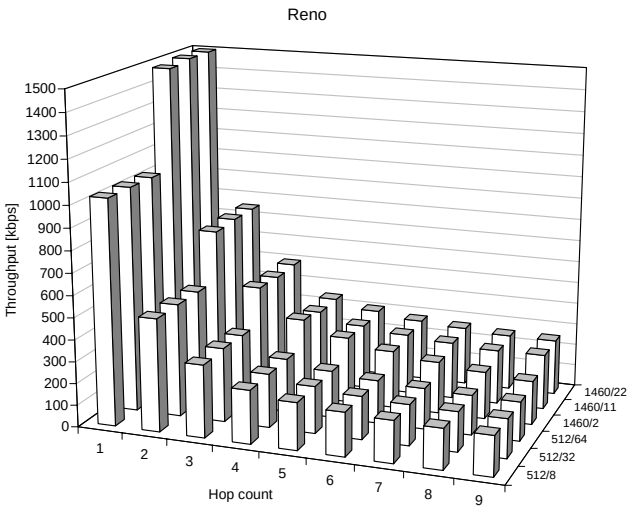

Figure 1: Throughput in linear network with the ZERO protocol

In order to derive T_i for all $i, 1 \leq i \leq 9$, a static network of two to ten nodes were set up, forming a linear chain of one to nine hops. The nodes were configured to run the ZERO protocol and a single 4 MB TCP transfer was performed. The reference protocol ZERO is configured statically with all the routing information needed to forward traffic before the simulation is started. Thus ZERO has zero routing and negligible forwarding overhead. This is analogous to the measurements made by M. Gerla et al. [22] but for a different MAC layer protocol and similar to the measurements by G. Holland and N. Vaidya [21] but for a larger set of TCP parameters. The T_i values for TCP/Reno are summarized in Figure 1. TCP/Reno and TCP/Sack performed about equal. TCP/Sack performed at most 18.8 % worse and at most 7.1 % better than TCP/Reno.

In Figure 1 when the hop length doubles, the throughput is halved. This is an effect of the IEEE 802.11 solution to the hidden station problem in combination with TCP window management. The rule is when a node overhears either a RTS or CTS it must be silent. This makes the network between node i and node $i+1$ and the network between node $i+1$ and node $i+2$ into a single channel. When one of the three nodes is ready to send, all of the others must remain silent. Node i can contend and win channel allocation only for a limited amount of tries until it reaches the maximum amount of outstanding packets allowed by TCP. At this point it has to wait until node $i+1$ sends all of temporarily stored segments to node $i+2$. Before node i can advance its window further the acknowledgement must be received. However, the throughput is halved when the hop count increases to a certain number. When it grows larger than five, some parallelism is introduced. Data can be

sent or received from both ends of the chain simultaneously.

Assume that we have a linear network as described in the preceding paragraph. Two factors then limit the throughput of TCP connections. The first and is the capacity of the link, C, the other is the maximum window size W. When W is large enough, the throughput is only limited by the capacity of the link. However, when W is the limiting factor, the minimum window size to fully utilize the bandwith (or throughput) B given a round-trip time RTT is given by [19]:

$$W = B * RTT$$

The throughput for a connection given the link capacity, the maximum window and the round-trip time is expressed as:

$$B(W) = \min(C, \frac{W}{RTT})$$

Assume that we have a connection and we send D amount of data. In a linear network of length $n \leq 5$ the transfer time is proportional to the number of hops in the network. If the number of hops doubles the transfer time also doubles. This is due to the single-channel effect. Sending D amount of data takes t_k seconds when the hop length is k and $2t_k$ when the hop length is $2k$. The throughput is then halved since:

$$B_{2k}(W) = \frac{D}{t_{2k}} = \frac{\frac{D}{t_k}}{2} = \frac{B_k(W)}{2} \tag{2}$$

In Figure 1, comparing the message sizes, we see that the smaller the segments size, the lower the throughput will be. This is because of the fact that TCP/IP and the MAC layer adds byte overhead to each segment. The overhead comes from the the TCP and IP headers together with the MAC header. Since the overhead is constant for each packet and independent of segment size, more packets means more overhead. The more overhead the longer overall transmission time and thus lower throughput. Also, there is not a great difference in performance in the two TCP implementations. The TCP/Sack algorithm seems to have a somewhat lower performance, probably due to the TCP option causing an extra packet overhead of up to 40 bytes in the acknowledgement.

7.1.2 Measured Throughput

Comparing the measured throughput in an experiment and the expected throughput, they may not be equal for several reasons. First, expected throughput does not take the routing overhead into account. Most of the routing

Stream	Reno	Sack
One	0.518	0.521
Two	0.514	0.509

Table 1: Average fraction of expected throughput

overhead comes from reconfiguration procedures when a link fails due to mobility. Also, in the case of DSR, there is a packet overhead in the data segments due to the enclosed route. This overhead increases the transmission time, which leads to an increased round-trip time and the throughput is decreased. Second, the routing protocols may not even find the shortest path between sender and the receiver. In a scenario when the routing protocol picks a non optimal route the measured throughput will be lower than the expected throughput. Finally, in a network with several simultaneous TCP transfers, the throughput will drop considerably due to the MAC layer RTS/CTS exchange. Although two transfers may use two completely separate nodes, they may interfere with each other. The closest two paralell transfers can come without interfering with each other is if the next hop node, as well as the current node, are at least two wireless hops away from each other. Otherwise, if the network contains fewer hops, it will turn into a single channel and the throughput drops.

In a situation where two transfers, using the same segment size, share a section turned into a single channel, the bandwidth on this section drops to a half. Assuming the node at one end of this channel picks a packet to send from each stream every other time, then a segment has to wait twice as long to get across the common section.

To verify this intuition, a static network of two to ten nodes were set up, forming a linear chain of one to nine hops. The nodes were configured to run the ZERO protocol and two 4 MB TCP transfer were performed. Each transfer start at the same node and end up at the same sink, and share the entire path. Table 1 summarizes this experiment. This table shows the fraction of expected throughout each stream obtained for a particular set of TCP parameters averaged over all path lengths. In some scenarios the first segments of stream two is lost due to limitations of ARP buffering. This shortcoming causes to TCP halt operation until its timer expires, which sometimes is as much as six seconds. This explains that for some scenarios, the achieved throughput is other than half of the expected throughput.

With a similar argument as above it can be showed that the throughput is inversely proportional to the number of parallel transfers. When the there are three transfers the throughput drops to a third, when there are four, it drops to a fourth et cetera.

7.2 Loss Rate

The number of segments dropped by the network is an indicator of the aggressiveness of a TCP implementation. Also, since the simulator does not support transmission errors, the number of lost segments is an indicator of how well the routing protocol is able to mask node mobility and link changes for TCP. The loss rate is correlated to the throughput of a connection. A high loss rate causes TCP to do time-outs and retransmissions of data more often lowering performance.

The throughput of a TCP/Reno connection can be estimated given the loss probability. Let p be the probability that a packet is lost and let RTT be the average round-trip time. Furthermore, let W_{max} be the maximum size of the congestion window and T_0 the amount of time the sender waits on a time-out. Then the the amount of packets transmitted by a sender per time interval, or the *anticipated* throughput $B(p)$, with unlimited amount of data is [20]:

$$B(p) \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{4p}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3p}{4}}\right) p(1+32p^2)}\right) \qquad (3)$$

In this thesis the loss probability p is approximated by the number of retransmitted TCP data packets over the total number of TCP data packets sent. The approximation of loss probability is identical to the one used in [20]. The consequence of approximating the loss rate in this way is that it does not include the possibility that an acknowledgement is lost. Also, it indicates a loss which is higher than the actual loss of packets in the network. In a MANET the cause of a TCP retransmission may be very complex and not necessarily due to packet loss. The RTT and the T_0 are calculated as the arithmetic mean of corresponding instantaneous values over the lifetime of the connection.

Figure 2 shows the ratio of the packets transmitted and the value given by Equation 3 for a linear network of varied hop length routed with the ZERO protocol. The ratio was calculated both for TCP/Reno and TCP/Sack even though TCP/Sack violates the assumptions of this formula. However, the values for TCP/Sack are not showed in Figure 3 since they were almost the same. The values were on average 2 % larger. For any TCP/parameter and hop length a ratio of one indicates that the formula was able to predict the throughput. In the case of TCP/Reno, all measured values roughly corresponds to their respective estimated values except for when the segment size is 512 Bytes and the maximum window is eight packets and when the segment size is 1460 Bytes and the maximum window is two packets. In these

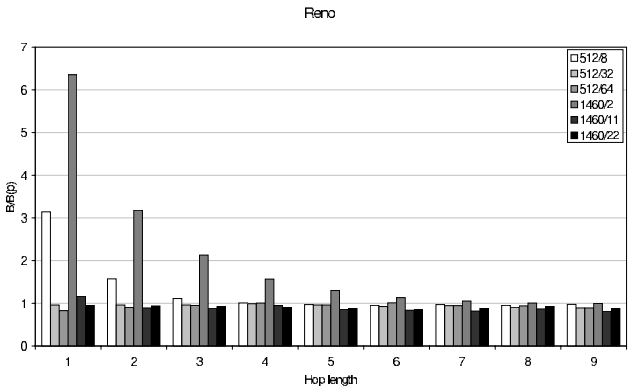

Figure 2: Loss rate/throughput correlation in linear network.

cases there is little or no loss and thus $p \approx 0$. The formula then fails to provide a reasonable estimate on the throughput since it is given by $\frac{W_{max}}{RTT}$ which is much less than the measured throughput.

7.3 Router Queue Length

The router queue length gives an indication of the end-to-end delay in the network seen by an application. In a bulk data transfer over a wired network, this may not matter much, but in a MANET, the router queue space is a bottle-neck resource. Althoug a node in a MANET is a router they are not envisioned to support large router queues. When router queues are small even a moderate network traffic will cause som nodes router queues to overflow.

The capacity of a wireless link is often lower than in a wired network [2], less data will be transferred per unit of time and more data will build up on each node. If TCP sends packets aggressively, congestion will soon build up, lowering performance. Also, packets should not be kept at intermediate nodes for long periods of time. If there is a topology change in the network, packets in an intermediate node may be lost or reordered. The lost data will cause retransmissions, but reordered packets may not, depending on the version of TCP used.

The router queue length is measured before a packet is enqueued. No attention is payed to which sub queue the packet is assigned. This means that the lowest number of packets in a queue reported is zero and that the measurements are spread unevenly in time. The router queue length for each node is approximated by the sum of the weighted queue lengths. A weight is the fraction of the time a queue experienced a particular queue length over the total time a queue has been in use.

8 Experiments

The experiments are divided in three batches. The first batch consists of the *One-to-four Hop* experiment and the *Overlap* experiment. The overall purpose of the first set of experiments is to explore the fundamental mechanism of the simulated MANET environment. Primarily, these experiments are designed to to be easy to verify for correctness with respect to the protocol specifications. Secondarily, the aim is to measure TCP performance. The One-to-four hop experiment consist of a single TCP transfer in a MANET with low mobility and few route changes. The overlap experiment puts a higher load on the network since two TCP transfers are performed.

The second batch of experiments is designed to give a thorough understanding of what parameters effect TCP performance. The second batch consists of two *Random way-point* experiments. Both experiments have in common that the nodes move about randomly. For each transfer performed, a source and a destination is picked at random as well. The purpose of the experiment is to obtain non-biased values for each metric. In the first experiment a single TCP transfer is performed, the second two identical TCP transfers are performed.

Finally a third batch of experiments will be presented. The *Realistic* experiment is designed to test application level generated TCP traffic in a MANET. The traffic pattern used is derived from a statistical World Wide Web model. The traffic pattern is tested in a network where the nodes have a random movement pattern.

The communication pattern parameters for all experiments are: TCP version, routing protocol, TCP maximum window size, TCP segment size and the number of TCP transfers. Two TCP versions are compared for performance, TCP/Reno and TCP/Sack. TCP/Reno contains the four basic mechanisms, congestion avoidance and control as well as fast recovery and retransmit. TCP/Sack uses also the selective acknowledgement option. These two TCP versions are run in MANETs routed with two protocols, DSR and AODV. DSR is configured to use the promiscuous mode option and AODV is set to use link layer notifications. The TCP implementations were run on a set of segment sizes, 512 and 1460 Bytes. They are the most common sizes of segments exchanged on the Internet [23]. The segment size remains the same throughout the lifetime of a connection. Finally, the experiments were tested with maximum window sizes of 4 kByte, 16 kByte, and 32 kByte. These values were chosen since they are the most common values found in system implementations restricting the window size [14, page 252]. However, the simulator is packet oriented and the maximum window governs the maximum number of outstanding packets. The experiments are performed for

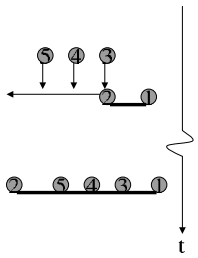

Figure 3: Mobility scenario.

window size of 8, 32, and 64 packets, when the segment size is 512 Bytes. When the segment size is 1460 Bytes the window is 2, 11, and 22 packets.

8.1 One-to-four Hop Experiment

The purpose of this scenario is to study the performance of a single TCP transfer in a MANET with relative low mobility. The mobility scenario consist of a linear chain of nodes forming a network with increasing hop count.

The mobility pattern of each node is summarized in Figure 3. Due to the mobility and the radio transmitter range is 250 m, node connectivity in the network will change at 5, 25, 45 s after the simulation starts. Node one and node two are set up take take part in the TCP transfer. The hop count on the shortest path between node one and node two is one until the first connectivity change, then two, then three and finally four after 45 s of simulation run. In this scenario, a file transfer of exactly 4 MB starts as soon as the simulation starts - at time zero. It is sourced at node one and sinked at node two.

8.1.1 Measured Throughput

The measured throughput for the one to four hop scenarios are summarized in Figure 4. The measured throughput in the scenarios running DSR are close to their corresponding expected throughput values, despite the packet overhead. The promiscuous mode prepared the nodes with alternative routes well in advance before the link fails. The extra cost of reconfiguring the network is kept to a minimum. The effect of different window sizes is not

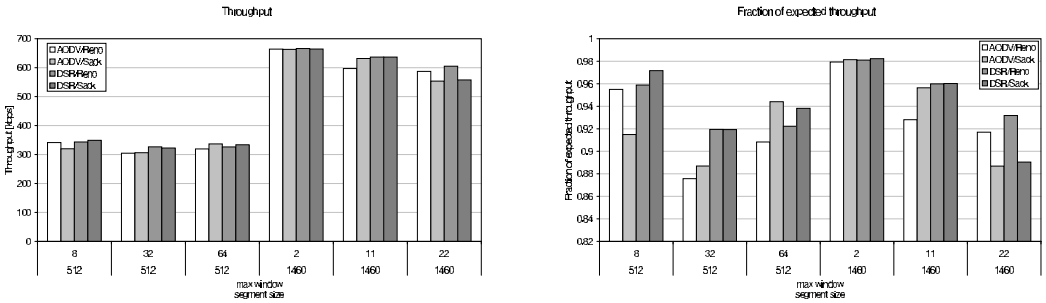

Figure 4: Throughput and fraction of expected throughput.

easy to perceive, but it seems to be slightly better to use a smaller window. DSR goes to great lengths to salvage packets, which means that packets sent on a failing link quickly can be rerouted. The salvaging ability of DSR only causes a longer delay rather than dropped segment and retransmissions.

Comparing the fraction of expected throughput for TCP/Reno routed with AODV and DSR we see that there seems to be a small advantage using DSR. The throughput achieved for all experiments using DSR is always better than about 92 percent of the expected throughput.

The scenarios using AODV suffers somewhat from timing problems. When using TCP/Reno and window size of 32 and in scenarios using TCP/Sack together with window sizes of eight, 32 and 22, a timing effect at the MAC layer causes a delay for 10 s. In the three first of these scenarios, what happens is that a MAC layer RTS message from node one to node three is garbled due to collision at every try. When node one finally times out, it reports the failure to AODV. AODV concludes the link is down and sends a RREQ. However this RREQ is also garbled and lost at node three. Node one now waits a total of 10 s before it sends the next RREQ.

A similar thing happens in the scenario using TCP/Sack with a window size of 11 but at a different node. In this scenario, node four is the node that does not get its RTS messages across. The 10 s delay in all of these scenarios explain the dip in the achieved fraction of expected throughput seen in Figure 4.

8.1.2 Measured Loss Rate

In Figure 5 we find the measured loss rate in the experiment using DSR and AODV. The loss rate increases with the window size. With more outstanding data, when a link failure happens there will be more packets that will either be lost or reordered. Lost packets will cause TCP to retransmit - reordered packets may cause retransmissions. There is no advantage using TCP/Sack

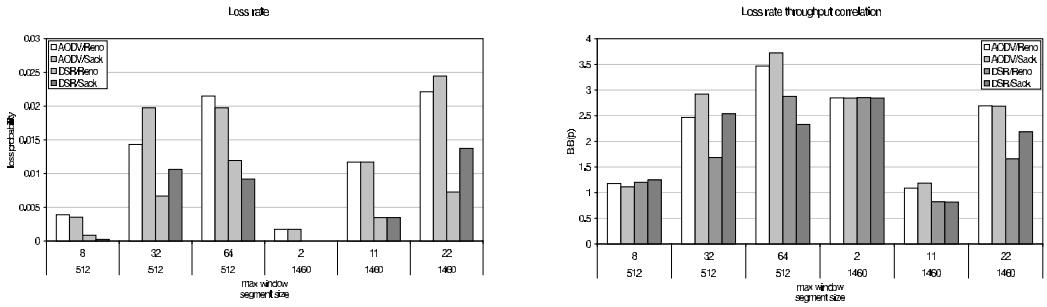

Figure 5: Loss rate and loss rate/throughput correlation.

over TCP/Reno. This is remarkable since TCP/Sack is designed to lower the number of retransmissions by sending only the missing packets. However, many of the retransmissions are actually caused by waiting for routing procedures to recover route changes, no packets are lost. For instance, in both routing protocols, a node may not rebroadcast a Route REQuest (RREQ) until a timer expires. The value of this timer is sometimes quite large, sometimes as high as about ten seconds in AODV. If the initial RREQ or the route reply is lost, TCP may time out several times. If a time out happens with a full window, the loss rate increases rapidly.

AODV is bad compared to DSR when it comes to loss rate. The loss rate measured when using AODV is several times higher than in a corresponding scenario routed with DSR. AODV has a stricter packet salvaging policy - it can only reroute and retransmit packets if they are found at the same node as they were originated.

There is a correlation between the loss rate and the fraction of expected throughput. A low loss rate yields a higher fraction of expected throughput. Comparing the loss rate and the fraction of expected throughput, DSR has an edge over AODV because DSR's loss rate is lower. However, the difference in measured throughput, comparing DSR and AODV is not very big. The high loss in AODV is compensated by a lower packet overhead. In DSR it is the other way around; a low loss rate, but a higher packet overhead.

There is no correlation between the loss rate and the throughput as anticipated by Equation 3. In Figure 5 for no parameter the measured values are within their corresponding anticipated values as given by this formula. The reason for this is the wide variety of loss causes that cause retransmission. The underlying model for Equation 3 is that packets are lost due to shortage of router buffer space. Furthermore, this model assumes that packet losses are correlated, which is not always the case in a MANET as simulated here.

In Figure 6 the loss causes and the amount of packets lost due to these causes in scenarios using AODV is summarized. A similar summary but for

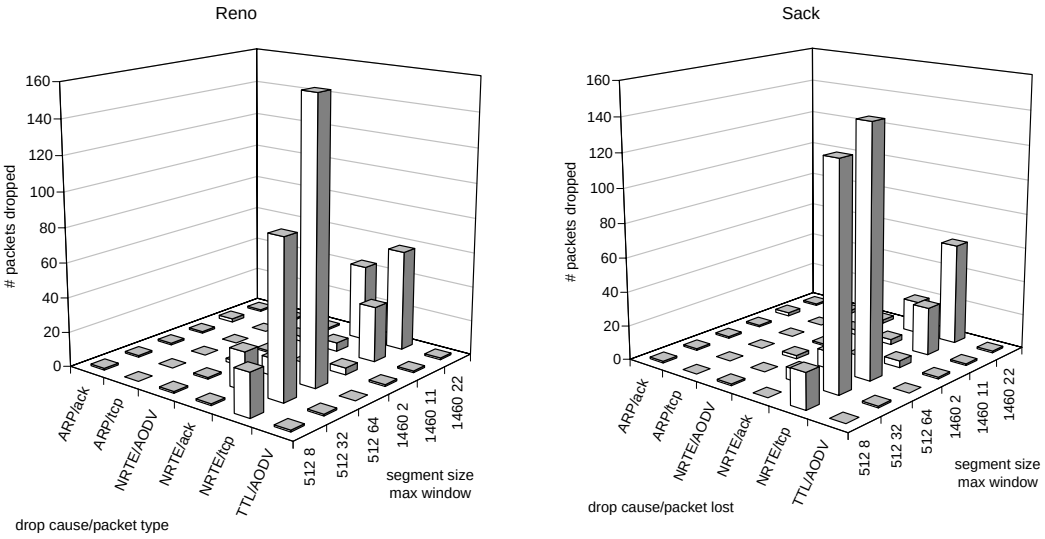

Figure 6: Packets dropped using AODV.

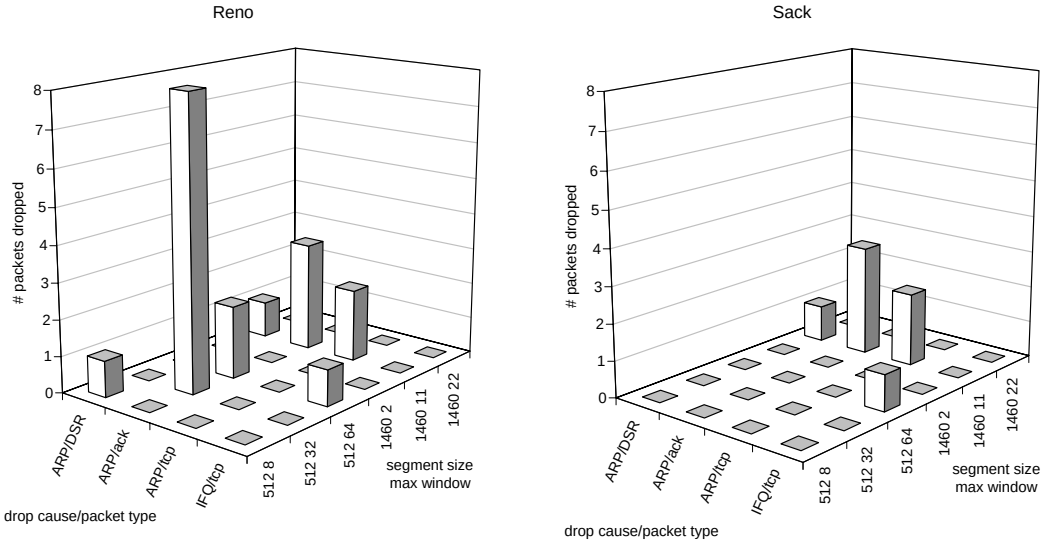

Figure 7: Packets dropped using DSR.

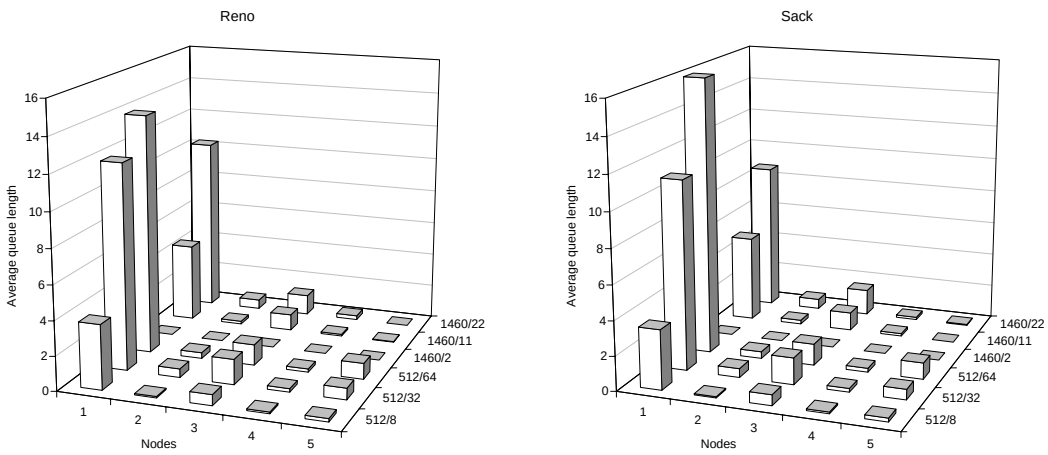

Figure 8: Average queue length using AODV.

DSR is shown in Figure 7. An explanation to a particular loss cause code and packet type can be found in Appendix B. AODV and DSR has entirely different loss cause patterns. AODV drops a lot of packets in the process of reconfiguring the network on a link change. When AODV is in a situation where it has no route for a particular destination it drops all packets going to this destination. The amount of TCP data packets dropped increases with window size. There is a slight difference in TCP version. Sack seems to cause AODV to drop less packets. In DSR however, not many packets are dropped at all. A disturbing fact is that DSR seems to loose a lot of packets in the ARP buffer. This causes a longer delay than necessary when reconfiguring the network. Comparing TCP version, TCP/Sack yields a lower loss of packets.

8.1.3 Measured Router Queue Length

The average queue length per node, for each node and parameter, is shown in Figure 9 for a network routed with DSR. There is no obvious difference in scenarios using TCP/Sack and scenarios using TCP/Reno. In Figure 8 the average queue length per node in a network routed with AODV is shown. No obvious difference using TCP/Reno and TCP/Sack is found.

In all scenarios, node one is always the most loaded. The gap in queue length, between node one and the second most loaded node is very large. In some scenarios node one's queue is almost five times larger than any other nodes queue. We have that the path length is low and the 802.11 solution to the hidden station problem turns the entire path into a single channel. When a packet is about to be transmitted from any node along the path, all other must remain silent. Also, due to the back-off mechanism in the contention phase, a node that has sent a packet must pick a new random wait time, but

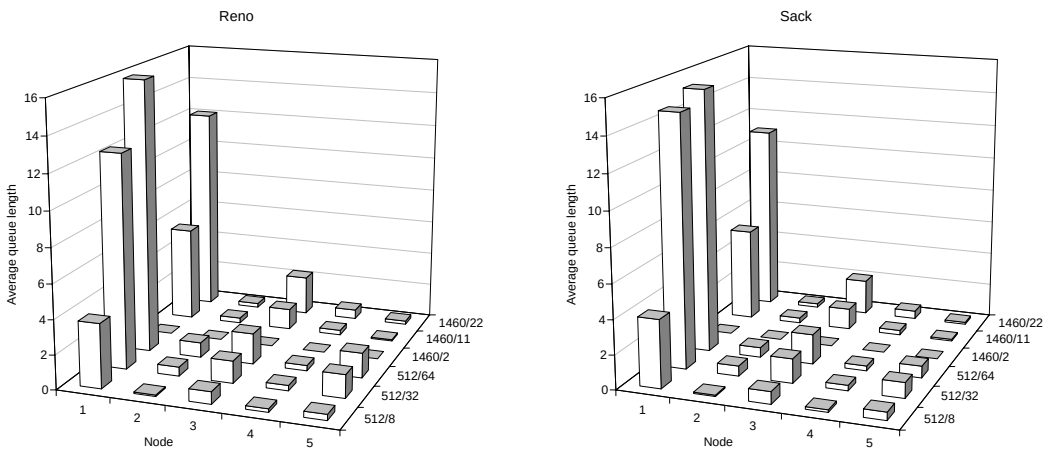

Figure 9: Average queue length using DSR.

all the others decrease theirs. This means that the possibility that the node that recently won will win the contention again decreases. In this situation the queue in node one will build up since the amount of packets coming in is far higher than the amount it is able to transmitt.

In the simulator the TCP data segments and acknowledgements have different priority which makes the end-to-end delay asymmetric. Acknowledgements travel fast and data slow. Under these circumstances, the TCP transmission strategy will fill the interface queue with data packets at node one. The transmission strategy works on the assumption that intermediate nodes can receive and forward packets at the same time, which is true in a non-shared wired network.

8.2 Overlap Experiment

The aim of this experiment is to examine how the MANET environment handles parallel TCP streams. The experiment consists of a small number of nodes under low mobility. Two source and sink pairs are set up and a transfer is performed between each pair. The nodes are set up to form a network where the transfers first cross each other over a single node. As simulation progresses, the transfers overlap, sharing a common wireless hop.

The experiment consist of five nodes. The mobility pattern is summarized in Figure 10. Given the mobility pattern and the range of the radio is set to 250 m, we have three connectivity changes. The changes happens at about 17, 37 and 47 s of simulated time. Each connectivity change forces at least one of the streams to change route. In Table 2 the shortest path between each source and sink is listed. Two TCP file transfers of 4 MB each are performed

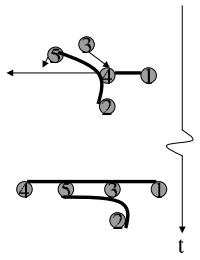

Figure 10: Mobility scenario

Time	one	two
0-17	1→4	2→4→5
17-37	1→3→4	2→4→5
37-47	1→3→4	2→3→5
47-	1→3→5→4	2→3→5

Table 2: Shortest path in each TCP stream.

simultaneously. The first is sourced at node one and sinked at node four. The other stream is sourced at node two and sinked at node five.

8.2.1 Measured Throughput

Figure 11 summarizes the measured throughput for each stream, routing protocol and TCP parameter. Comparing the throughput obtained by using different routing protocols we see that the difference in each protocol is small. However, comparing the throughput obtained using AODV and DSR in stream one, we see that DSR performs a little worse. In all the scenarios there is a lot of network traffic, all data passes through a collection of nodes, within a radius of one wireless hop. Since the RTS/CTS exchange forces the whole network into one channel and the frames transmitted are large, the link layer procedure takes longer time. A node retransmits a RTS several times before reporting a link is broken. Since there is a lot of contention with many nodes wanting to send long frames, the total time it takes to win channel allocation and re-send the RTS is large. On top of this, DSR adds packet overhead which adds to the time it takes to transmit a packet and therefore

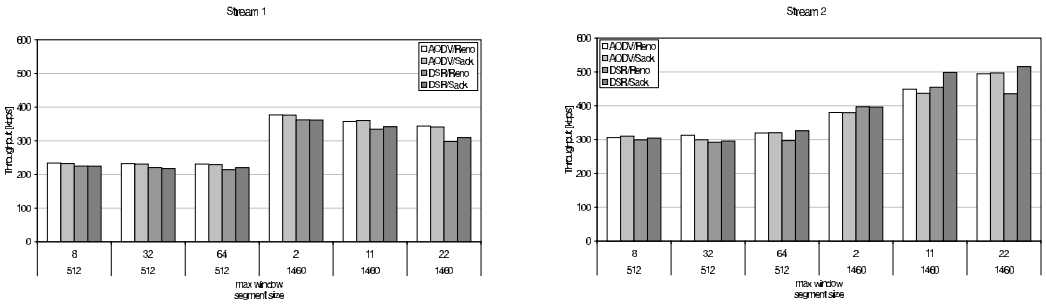

Figure 11: Throughput

DSR performs a little worse than AODV.

The gap in throughput comparing the two streams is explained by that they have different paths. The difference comes from that they spend an unequal amount of time at a certain path length and that they have different amount of route changes. Looking in Table 2 we see that the first stream has spends more time at a longer path. The path in stream one shifts from one to three hops. The second stream has a two hop route throughout the experiment. Theoretically the difference in expected throughput for this scenario is calculated as follows. Given Equation 1 we pick a t so that $t_3 = t - t_2 - t_1$ and $t_3 > 0$. The difference in expected throughput for stream one and two is:

$$B_1 - B_2 = \frac{t_1 T_1 + t_2 T_2 + t_3 T_3}{t} - \frac{t T_2}{t}$$

By Equation 2, $T_1 = 2T_2$ and $T_1 = 3T_3$:

$$= \frac{2t_1 T_2 + t_2 T_2 + \frac{2}{3} t_3 T_2}{t} - T_2 = T_2 \left(\frac{\frac{4t_1}{3} + \frac{t_2}{3} + \frac{2t}{3}}{t} - 1 \right)$$

From Table 2, insert the values for t_1 and t_2:

$$\approx T_2 \left(\frac{32}{t} - \frac{1}{3} \right)$$

Thus $B_1 > B_2$ when $t < 96$, $B_1 = B_2$ when $t = 96$ and $B_1 < B_2$ when $t > 96$. To transfer 4 MB of data at a throughput of T_2 with two parallel streams operating with segment size 512 takes about 134 s. The amount of time using a segment size of 1460 about 96 s. Thus, using a segment size of 512, $B_1 < B_2$. However, using a segment size of 1460, $B_1 \approx B_2$.

In Figure 12 the fraction of expected throughput is shown. Note that the second stream appears to perform better than the anticipated fraction of

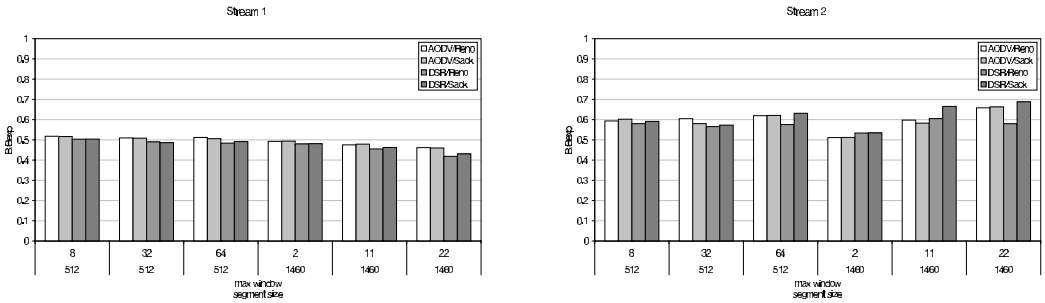

Figure 12: Fraction of expected throughput.

expected throughput, which is approximately 0.5. The second stream has an edge over the first, averaging at about 0.6 whereas the first stream performs at about 0.5 of the expected throughput for all parameters. The reason for this is that the first stream experiences two link failures while the other only has one. The routing procedure that take place to cover the loss causes a halt in the transfer. When there are no packets forwarded on behalf of the first stream, there is no contention for channel access for the other. We know that the expected throughput doubles when going from two parallel streams to one. Since the second stream does not experience the same halts as the first does, it finishes much earlier. When the second stream is finished the first has the channel for itself and it experiences a rise in throughput. This rise is high enough to cover for the loss it experienced earlier in the transfer and to obtain a higher fraction of the expected throughput.

Comparing expected throughput performance obtained by using different routing protocols, AODV is better than DSR for stream one. However, for stream two it is not so obvious. AODV and TCP/Reno performs better than DSR and TCP/Reno but AODV and TCP/Sack does worse than DSR and TCP/Sack.

8.2.2 Measured Loss Rate

In Figure 13 the loss rate for each stream is presented. The loss rate is higher for stream one than it is for stream two for all parameters. The reason is that stream two experiences a lower number of route changes. Fewer route changes means that loosing packets is less likely.

The loss rate seem somewhat correlated with the window size. A possible explanation is the greater the window size, the more packets will be outstanding and more packets will be lost on a route change. Further more, the loss rate explains the fraction of anticipated throughput obtained for stream two as seen in Figure 12. AODV and TCP/Sack has a high loss rate and

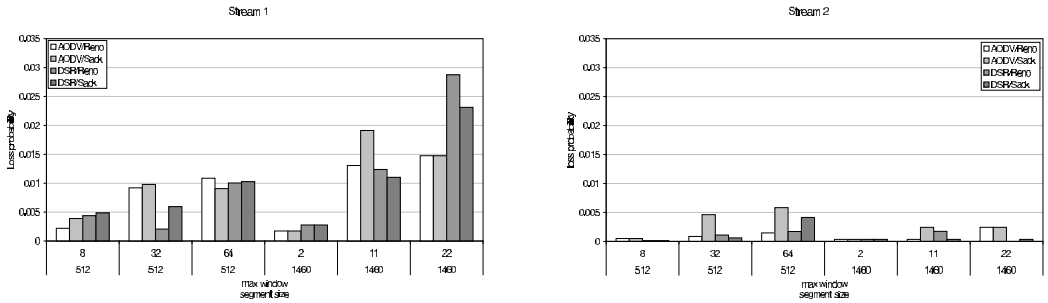

Figure 13: Loss rate.

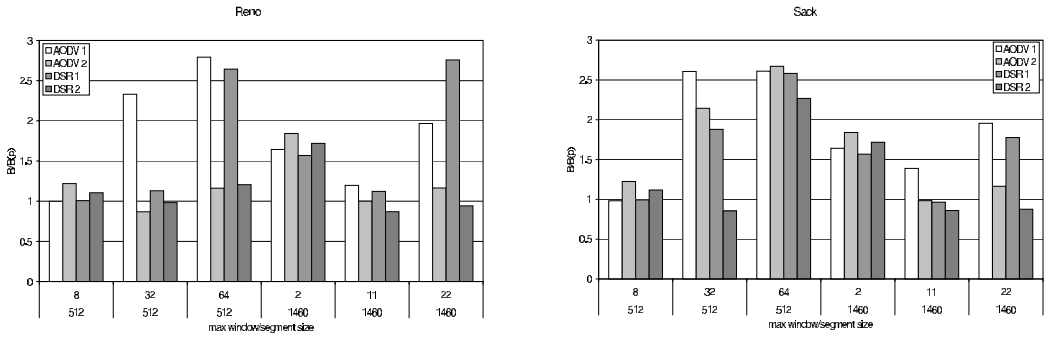

Figure 14: Loss rate/throughput correlation.

thus obtains a worse fraction of expected throughput compared to DSR and TCP/Sack.

The loss rate is not correlated to the anticipated throughput given by Equation 3. Figure 14 shows the fraction of measured and expected throughput. A perfect prediction of Equation 3 would yield a value of one. However, few of the values are close to one. As in the One-to-four Hop experiment, the reason for this is that there are many more loss causes in a MANET than the model assumes.

The loss causes for the scenarios using AODV and DSR are given in 15 and 16 respectively. The graphs show the number of packets dropped by both stream collectively. AODV drops far more packets than DSR. However the drop cause pattern is very varied comparing the protocols. AODV drops most packets due to link breakage where DSR drops none. A lot of AODV routing messages (actually Route Reply messages) are lost in the ARP buffer in many scenarios, however this phenomenon is not unique to AODV. DSR also causes drops of routing messages (also Route Replies) in the ARP buffer.

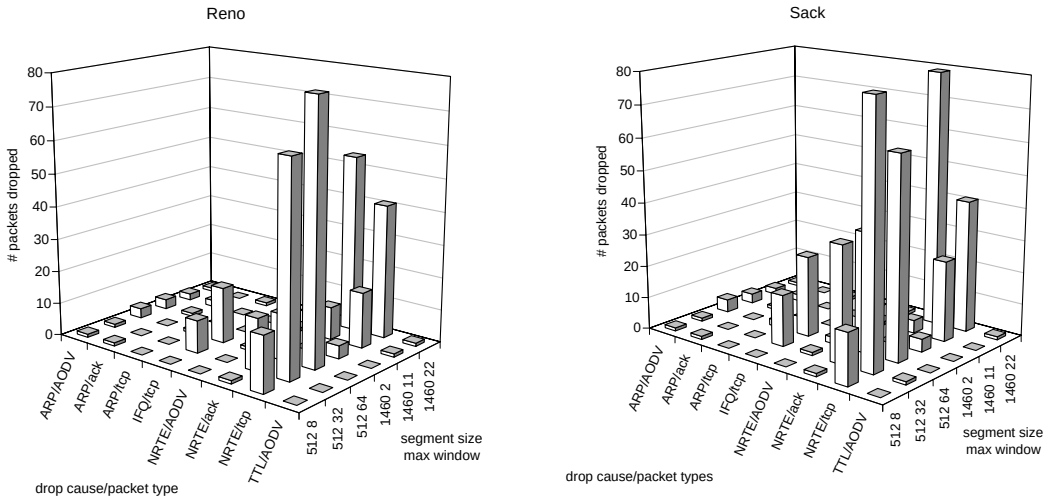

Figure 15: Packets dropped using AODV.

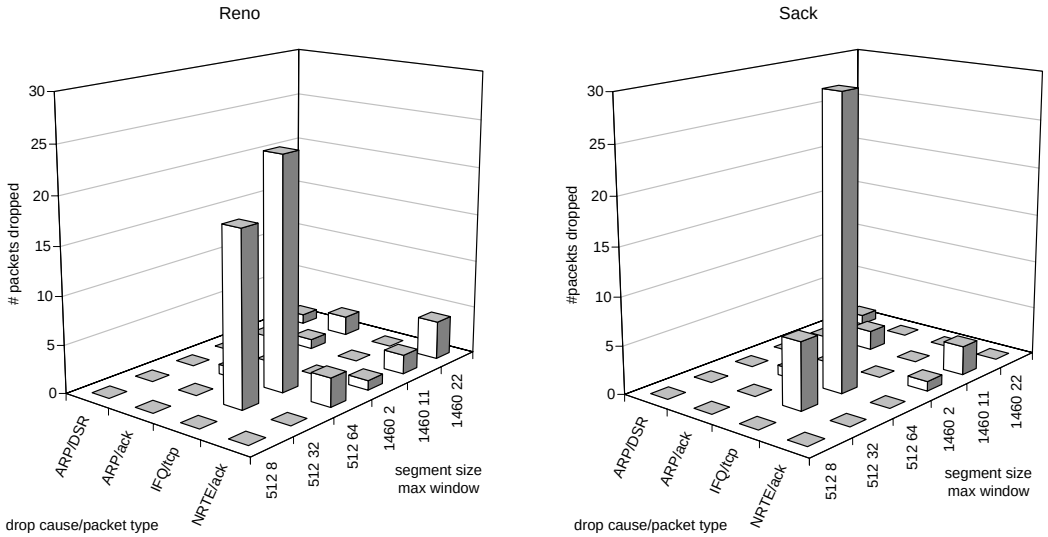

Figure 16: Packets dropped using DSR.

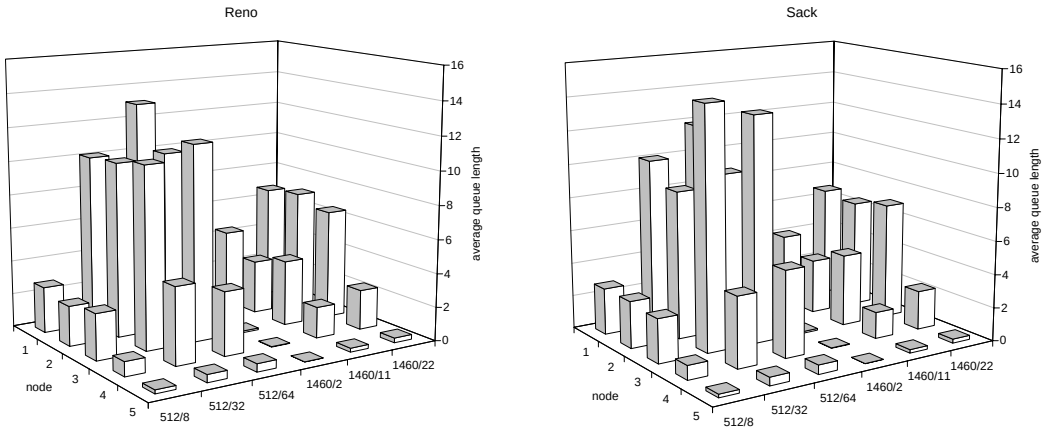

Figure 17: Average queue length using AODV.

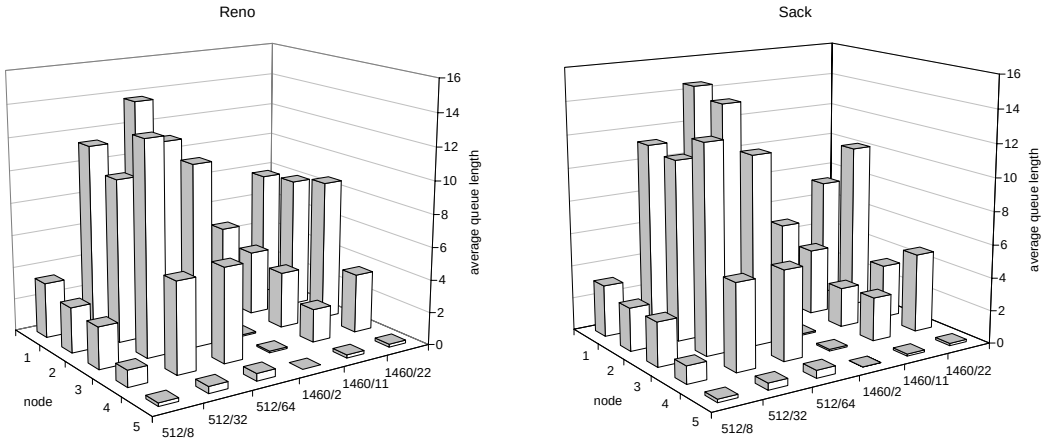

Figure 18: Average queue length using DSR.

8.2.3 Measured Router Queue Length

Figure 17 and 18 shows the average queue length in a network routed with DSR and AODV respectively. The source node in each stream is heavily loaded for similar reasons as in the One-to-four Hop experiment. However in most scenarios the interface queue in node three is the most loaded of all queues in the network. The reason is for this is that after 37 seconds of simulation time, node three will be on the path for both streams. In the scenarios with segment size 512 and window size of 32 and 64 both sources in both streams will feed node three with a large quantity of packets. The amount of packets accumulates and the maximum capacity of the queue is reached several times causing packet loss.

8.3 Summary

The first batch of experiments indicates that the path length is the main factor for TCP throughput performance. The throughput is inversely proportional to the path length, that is, the longer the path length the lower the throughput. Segment size has the second biggest impact on throughput. The larger the segment size the lower the byte overhead per packet and thus higher performance. Finally, loss rate has the least but still substantial impact on throughput. A high loss rate cause a lot of retransmissions resulting in bad throughput.

The loss rate co-varies with the window size, but the causes of retransmission may not be an actual packet loss. Retransmission in scenarios routed with AODV are caused mainly by the routing layer throwing away packets it cannot deliver on a link failure. DSR, on the other hand, does route salvaging, TCP packets are seldom thrown away. Both AODV and DSR suffers somewhat from a poor ARP buffer strategy. TCP acknowledgements, TCP data and routing message packets are being dropped.

Loss rate is not correlated to throughput as measured by Equation 3. The basic assumptions made to derive this formula is not valid in a MANET. The drops caused by router buffer overflow is only a small fraction of the total amount of packets dropped in either routing protocol. Also, packet loss is not correlated to each other. A packet may be lost, or reordered by link failure but following packets may not.

The failure of this model suggest that TCP itself is not well adapted to the MANET environment. TCP's flow control mechanism is based on the assumption that loss in the network comes from router overflow. The loss causes in a MANET are mainly due to node mobility and rarely due to router overflow.

Window size is the main factor for router queue length. For all window sizes and number of streams, packets build up in the queue at the source and at the destination but is generally low elsewhere in the network. The bigger the windows the longer the queues. In the case with two streams, where streams cross, packets build up proportional to window size. Given the low amount of packages lost due to router queue overflow this is not very telling metric.

8.4 Random Way-point Experiment

The aim of this experiment is to derive values of TCP performance. The scenario is designed to be neutral and not favor any aspect of the various protocols in a particular scenario. The experiments are evaluated for two

metrics, throughput and loss rate. Also the expected throughput is calculated and the loss causes are presented since they give good indication on the relative performance of each protocol. However, the loss/throughput correlation and router queue length is left out since they do not provide further understanding of TCP performance in MANETs.

The random way-point experiment is based on the network model used by G. Holland and N. Vadiya [21]. The model consists of 30 nodes moving around randomly in a 1500 by 300 m rectangular area. Nodes are configured to stop for 0 s upon reaching an immediate destination before embarking on the next. This means that the nodes are in constant motion throughout the scenario. The nodes are configured to a random speed v, where $v \in [0.9v_{mean}, 1.1v_{mean}]$ and v_{mean} is the mean speed. Two scenarios are tested for two values of v_{mean}, 2 m/s and 10 m/s. Routing protocol performance is sensitive to node mobility. 10 random scenario files was generated for each mean speed value, giving a total of 20 scenario files. The 2 m/s scenarios and 10 m/s differ not only in that the nodes travel at different mean speed, but they also have different movement patterns.

The communication patterns tested for each of these scenario files are divided up in two parts. In the first a single TCP transfer is performed, in the second two parallel TCP transfers are performed. The size of each transfer is 4 MB. For each transfer a source and destination is picked at random, but the same source destination pair is used in all tests over all values of TCP version, routing protocol, segment size, window size and scenario file.

8.4.1 Measured throughput - one stream

Figure 19 shows the measured throughput averaged over ten scenario files with nodes moving at the average speed of 2 m/s and 10 m/s. In the case where the mean speed is 2 m/s DSR has the advantage over AODV. However there is no conceivable difference between the two TCP implementations for both routing protocols. The main factor in performance in this set is the routing protocol.

The standard deviation of the throughput for AODV is about 260 kbps when using TCP/Reno with the segment size set to 512 Bytes and the mean speed is 2 m/s. Increasing the segment size to 1460 Bytes yeilds a standard deviation of about 390 kbps. When the mean speed is increased to 10 m/s the standard deviation is slightly lower, about 240 kbps for TCP/Reno and a segment size of 512. However, if the segment size is also increased, the standrad deviation is about 460 kbps. Regardless of routing protocol and TCP version, increasing the segment size from 512 to 1460 Bytes also increases the standard deviation of the throughput with about 40 to 50 % when the

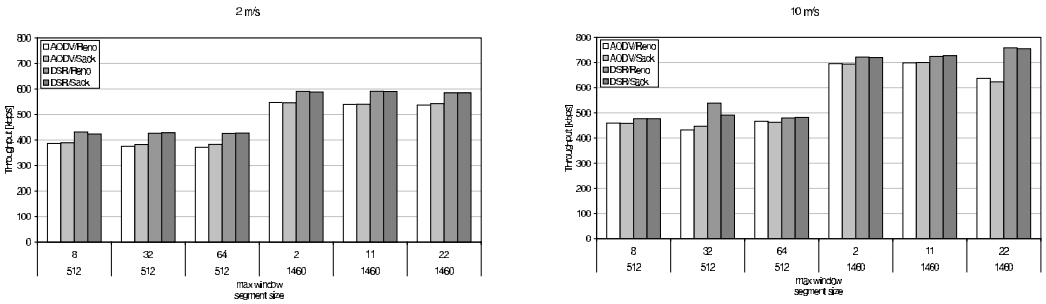

Figure 19: Throughput.

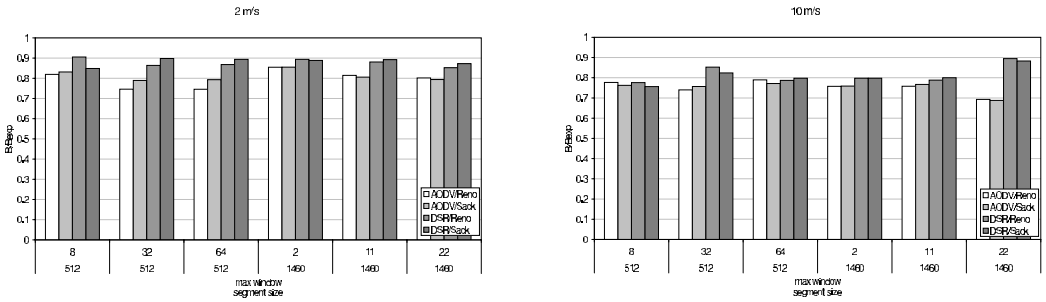

Figure 20: Expected throughput.

mean speed is 2 m/s. However, when the mean speed is 10 m/s, increasing the segment size increases the standard deviation with 75 to 100 %. AODV has a little lower standard deviation regardless of TCP version and mean speed. When the mean speed is 2 m/s the standard deviation is 10 % lower for AODV than for DSR. When the mean speed is 10 m/s standard deviation is about 7 % lower. Finally, using TCP/Reno over TCP/Sack lowers the standard deviation with about 0.5 %. This happens regardless of routing protocol and mean speed.

Comparing the throughput when the mean speed is 2 m/s and 10 m/s the throughput is generally higher in the 10 m/s case. This seems contradictory to what G. Holland and N. Vaidya obtained [21]. However, they used the same movement pattern for all mean speeds. This test is configured with different sets of movement patterns for each mean speed. The different movement patterns causes different paths and the path is the key factor in throughput performance. The optimal path length averaged over all scenarios is 2.7 for the 2 m/s scenarios and 2.5 for the 10 m/s scenarios. The average path length is lower in the 10 m/s case because with a higher average speed the nodes will be closer to each other more often.

In Figure 20 the fraction of expected throughput averaged over ten scenarios is presented. DSR performs a little better than AODV for both TCP

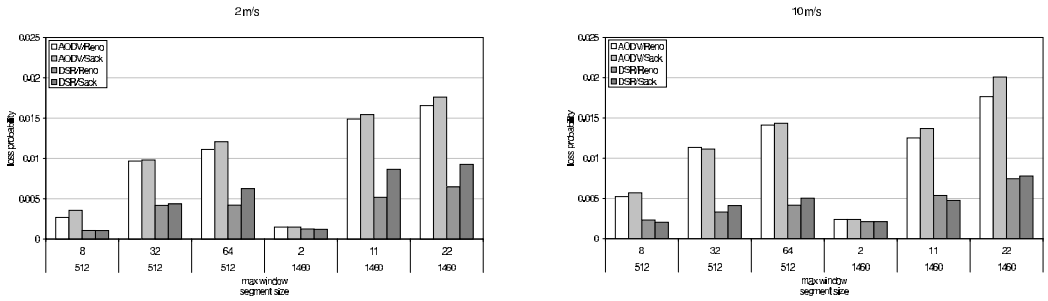

Figure 21: Loss rate.

versions in both mean speed cases. It seems that TCP/Sack performs a little better than TCP/Reno regardless of protocol when the mean speed is 2 m/s. However, when the mean speed is 10 m/s there is no difference in performance of the TCP versions. For all parameters, the fraction of expected throughput is higher in the 2 m/s case than in the 10 m/s case. This is because with higher speed the nodes will move apart more often, breaking routes more often which causes more routing overhead.

8.4.2 Measured Loss Rate - one stream

Figure 21 presents the loss probability averaged over ten scenarios. AODV has a generally a higher loss rate than DSR. This is because AODV drops more packets on route changes. Also, TCP/Sack generally causes a higher loss rate. Comparing the average values in the 2 m/s case and the 10 m/s case we see that AODV performs a little worse when the mean speed is increased. DSR, on the other hand is not affected by the increased speed. In some cases the loss rate decreases when the mean speed increases. The loss rate increases with the window size.

The loss causes in this scenario is routing protocol dependent. Figure 22 and 23 show the loss causes when the average speed is 2 m/s. The number of packets dropped are averaged over ten scenarios. AODV drops far more packets than DSR. However, the protocols have fundamentally different drop cause patterns. For instance, AODV drops a lot of packets due to TTL counter in the IP packet has reached zero. This is due to the route request flooding scheme used. DSR on the other hand looses a larger portion of its routing message (actually Route Replies) packets in the ARP buffer. Since AODV does not do packet salvaging, a large proportion of TCP data packets are lost on route failures. TCP acknowledgements and AODV routing messages also suffer this fate to a large extent. The amount of packets lost in this way is proportional to the window size. The more outstanding packets

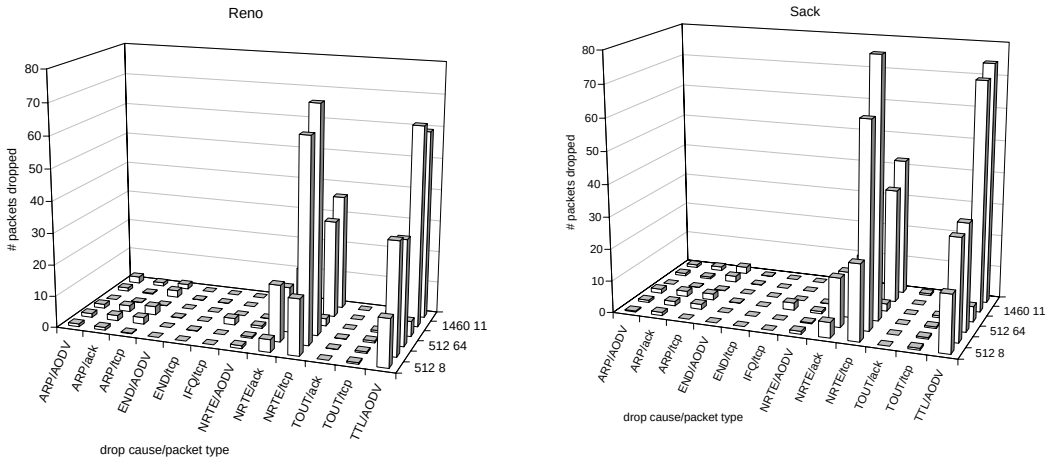

Figure 22: Packets dropped, average speed 2 m/s and AODV.

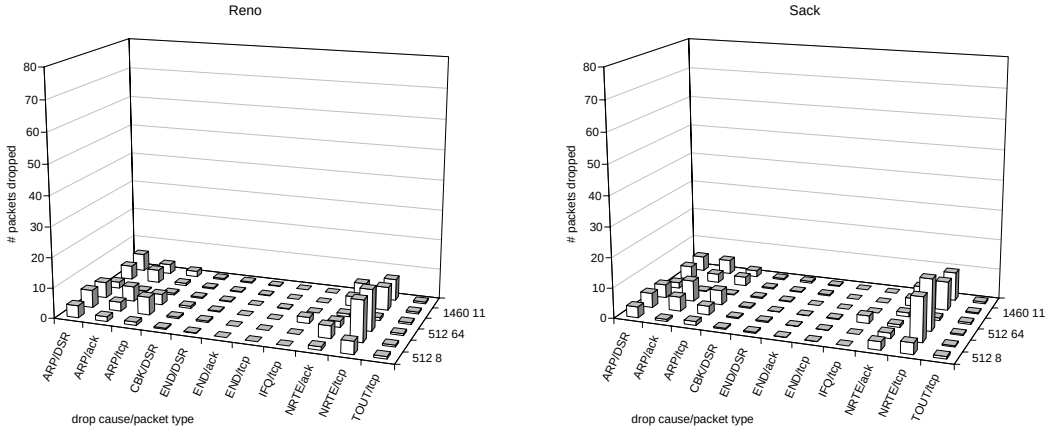

Figure 23: Packets dropped, average speed 2 m/s and DSR.

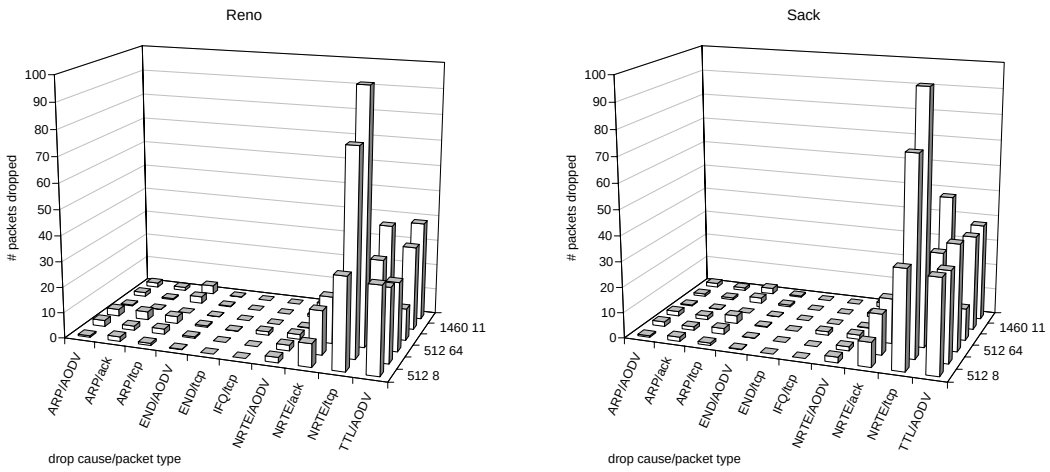

Figure 24: Packets dropped, average speed 10 m/s and AODV.

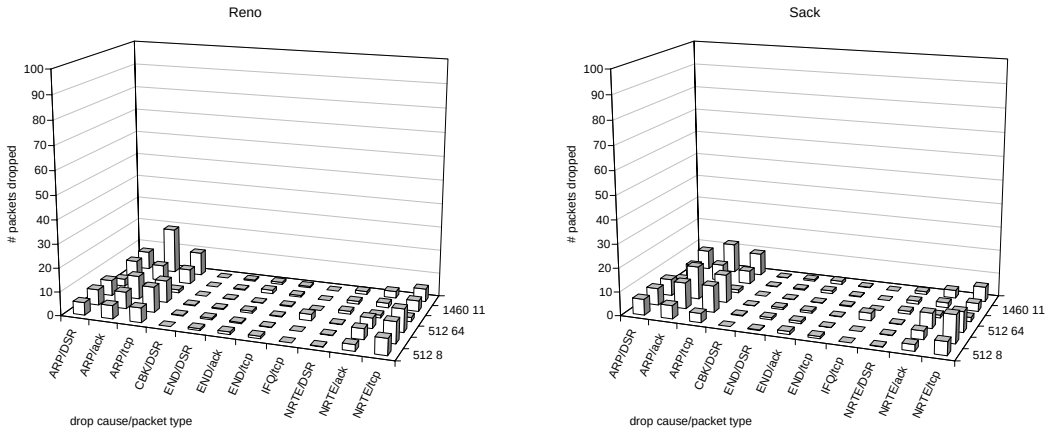

Figure 25: Packets dropped, average speed 10 m/s and DSR.

the more will get lost on a route failure. Few TCP data packets are thrown away due to buffer overflow. This only happens when the window size is 64 packets.

In Figure 24 and 25 the drop causes are showed for the scenarios with mean speed set to 10 m/s. The drop cause pattern for each routing protocol is basically the same. However, no packets are dropped due to TCP time outs in either protocol. This is because the networks spends less time in a state where a packet cannot be delivered. Generally the number of packets dropped for each cause is higher in the 10 m/s case than it is in the 2 m/s case. This is because the network spends more time reconfiguring than in a network with slower moving nodes.

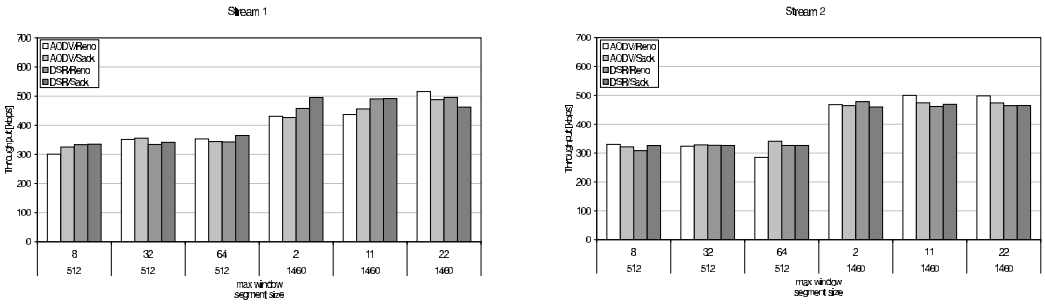

Figure 26: Throughput, average speed 2 m/s.

8.4.3 Measured throughput - two streams

Figure 26 and 27 summarizes the average throughput in the scenarios with an average speed of 2 m/s and 10 m/s when there are two streams active simultaneously. Comparing each stream against the single stream case they both behave about equal. The relative differences as observed in the single stream case is also present in the two-stream case with one exception. Going from a mean speed of 2 m/s to 10 m/s does not cause as steep increase in standard deviation as occured in the single-stream case.

However, comparing the averaged throughput obtained in this random way-point experiment and the throughput obtained in the random way-point experiment with only one stream, as shown in Figure 19, we see that the throughput is generally lower here. The throughput is about 100 kbps lower, comparing the one stream case with the two stream cases when the average speed is 2 m/s. In the 10 m/s case the difference is not as great. The first stream is only negatively effected when the segment size is 1460 Bytes. The throughput obtained here is about 100 kbps lower.

Finally, there is hardly any concievable difference in average throughput comparing stream one and stream two. Comparing the standard deviation for stream one and stream two, on the other hand, shows a difference. The standard deviation for stream one is almost always a little higher than it is for stream two, for all parameters. The standard deviation is at most about 51 % greater for stream one than it is for stream two.

Comparing the throughput obtained in the scenarios when the average speed set to 2 m/s and the scenarios where it is 10 m/s, the throughput is greater in the 10 m/s case. As in the single stream experiment, this comes from that the average hop length along the path from the source to the destination is lower in the 10 m/s case. Greater node velocity makes the probability that the nodes spend long period of times at a certain hop length away from each other less likely. Also, given the rectangular nature of the area

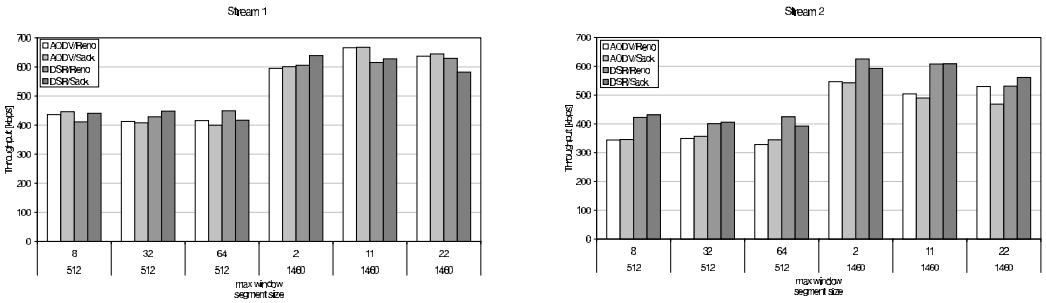

Figure 27: Throughput, average speed 10 m/s.

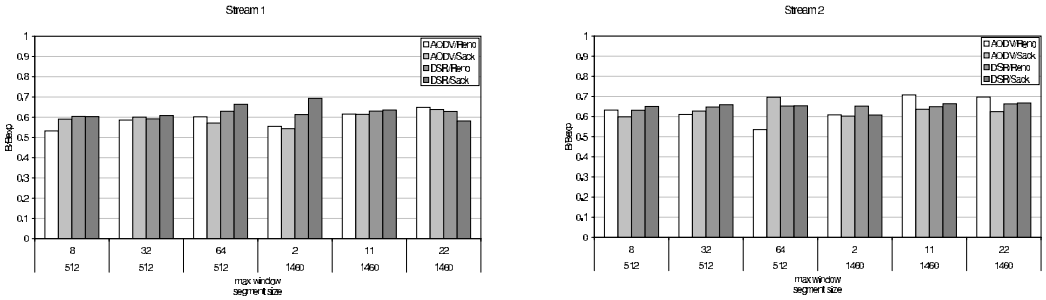

Figure 28: Expected throughput, average speed 2 m/s.

where the nodes move about, the probability of longer routes are unlikely. Taken these two facts collectively the average route length drops as the mean speed increases, simply because greater node velocity makes the time that two nodes spend at greater hop counts away from each other less common.

Looking at the fraction of expected throughput shown in Figure 28 and 29, we see that it is generally less than what is obtained in the random way-point experiment with one stream. The fraction of expected throughput obtained when the mean speed is 2 m/s is 25 to 30 percent lower than in the corresponding single stream experiment. In the 10 m/s case the fraction

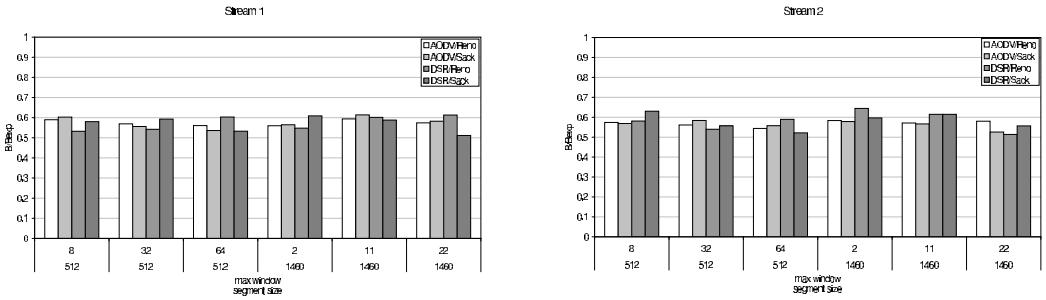

Figure 29: Expected throughput, average speed 10 m/s.

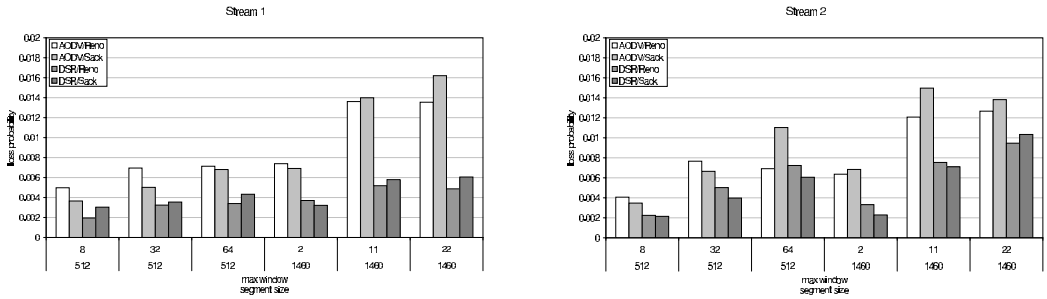

Figure 30: Loss rate, average speed 2 m/s.

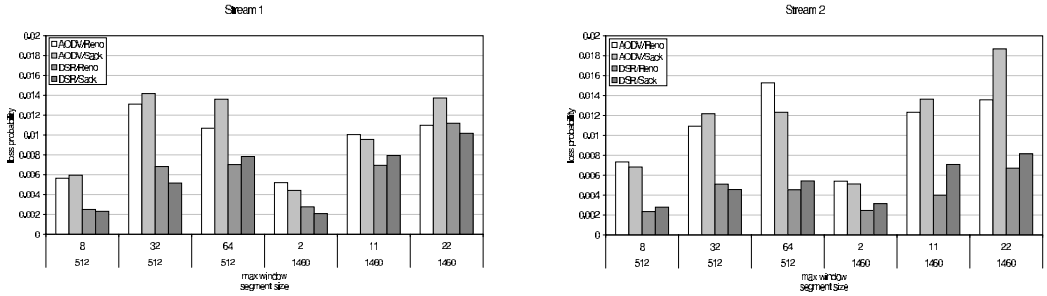

Figure 31: Loss rate, average speed 10 m/s.

of expected throughput is 24 to 30 percent lower. The extra contention for the medium caused by a second stream is the cause for this. It is however surprising the effect is this severe in a random way-point movement pattern. The average optimal path length when the mean speed is 2 m/s is about 2.5 and when then mean speed is 10 m/s about 2.2 for both streams. Since node density is fairly high, it seems unlikely length that the streams should interfere with each other. However, due to the RTS/CTS exchange, if two nodes comes closer than two wireless hops the they will interfere with each other. The scenarios consist of a 1500 by 300 rectangular area. Since the transmitter range is 250 m, it is, in fact, very likely that two parallel streams with an hop count of two or more will interfere with each other.

8.4.4 Measured Loss Rate - two streams

The loss rate is given in Figure 30 and 31. The loss rate is averaged over ten scenarios. For both mean speeds AODV has a higher loss rate than DSR. Comparing runs with mean speed set to 2 m/s and runs with means speed 10 m/s, the tendency is that loss rate is higher for runs with mean speed set to 10 m/s.

Comparing the loss rate in the single stream experiment and the loss rate obtained for both of the streams in the two-stream case when the means speed

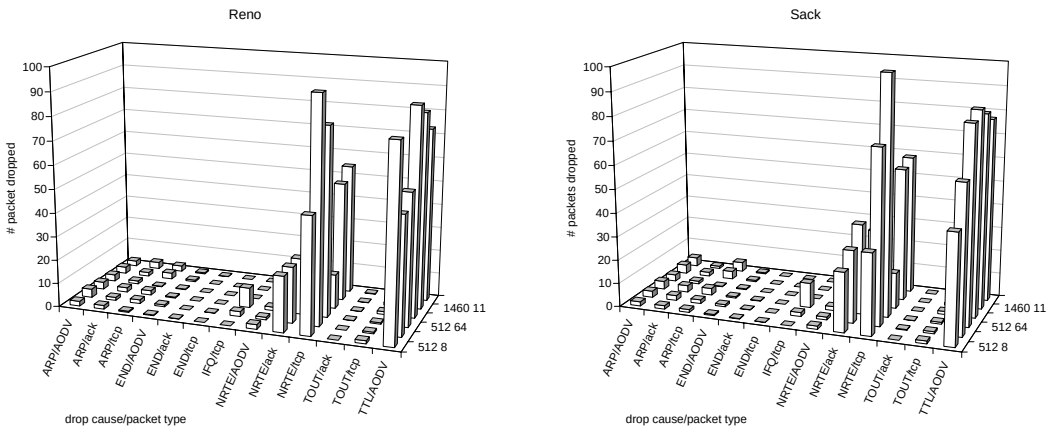

Figure 32: Packets dropped, average speed 2 m/s and AODV.

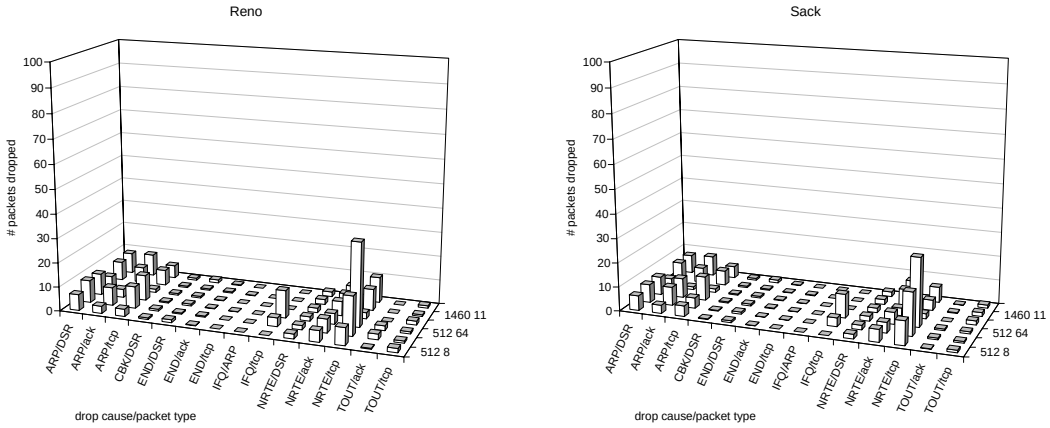

Figure 33: Packets dropped, average speed 2 m/s and DSR.

is 10 m/s, the loss rate is a little higher for both streams in the two-stream case. On the other hand, when the means speed is 2 m/s the two-stream case does a little worse than the single-stream case. However, neither of these differences are great and can probably be explained by the that each case has a unique movement pattern. The difference in loss rate may lie within the variance of movement patterns and may not be a function of the number of paralell streams.

The loss causes in the network when the mean speed is set to 2 m/s is shown in Figure 32 and 33. Both streams contribute to the number of packets dropped collectively. Comparing the single stream experiment with this experiment we see that the number of packets dropped for each drop cause only increases by a moderate number of packets. Comparing the scenarios using AODV the largest impact of having two streams is in the number of AODV

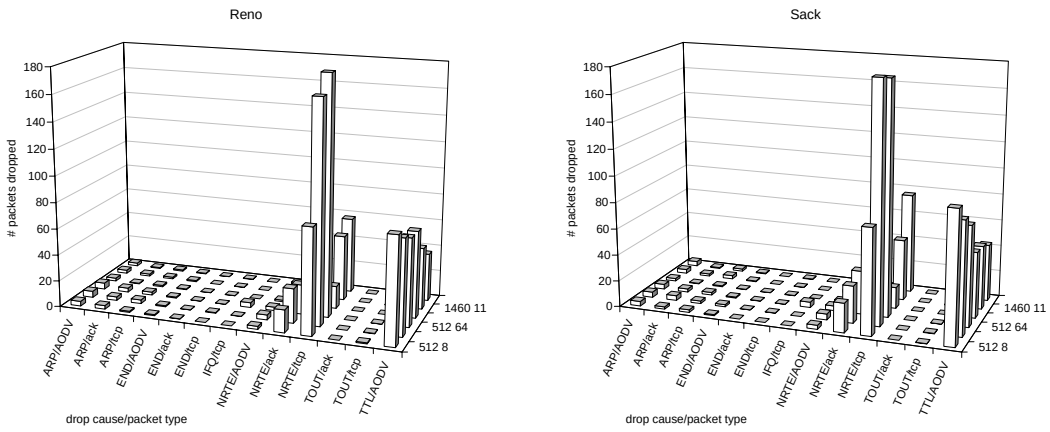

Figure 34: Packets dropped, average speed 10 m/s and AODV.

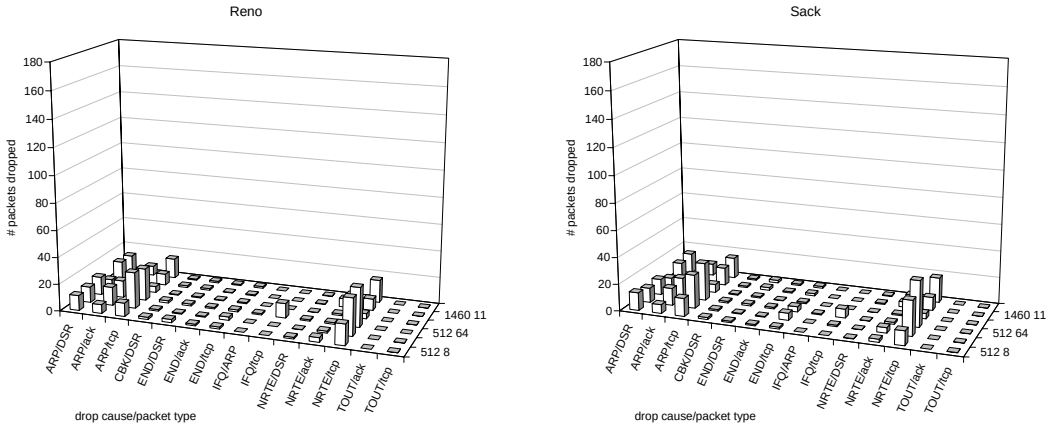

Figure 35: Packets dropped, average speed 10 m/s and DSR.

routing packets dropped when the segment size is 512 Bytes. The increase here is about 20 packets. However, for DSR the increase in the number of packets dropped for each drop cause is negligible.

The loss causes and the number of packets dropped for each loss cause and packet type when the mean speed is 10 m/s is shown in Figure 34 and 35. For the scenarios routed with AODV there is a moderate increase of packets dropped compared to the single stream experiment. Also, the 10 m/s case has a higher amount of packets dropped due to route failures. However, the drop causes and the number of packets dropped for each cause is very scenario dependent. A different source/destination pair in an otherwise identical network give rise to different drop patterns. The number of packets dropped in the ARP buffer is lower than in the single stream experiment.

8.5 Summary

The second batch of experiments gives that the path length is the main factor in TCP throughput performance. A low path length will give a high throughput. The choice of segment size has the second largest impact. A smaller segment size will decrease performance as the packet byte overhead increases. The choice of routing protocol has the least but still substantial impact on performance. AODV causes more packet drops than DSR, but gains extra performance due to the lower byte overhead per packet. The result is that the protocols performs about equal on average. However, there is a tendency that in a network with a lower route change rate AODV gives better performance due to the low packet overhead. In a network with higher amount of route changes DSR gives better throughput performance due to package salvaging. There is no difference in throughput using TCP version. No extra gain is provided using the TCP selective acknowledgment option.

The fundamental setback to TCP performance is the variety of loss causes in the MANET environment which disturbs the TCP flow control mechanisms. Only a small fraction of the loss causes are due to router buffer overflow. In the random way-point experiments, the window size did not affect TCP throughput much. The cause is that route changes that cause time outs are frequent. A TCP sender is rarely able to have a maximum number of outstanding packets.

The loss rate is protocol dependent. AODV gives a higher loss rate than DSR. The reason for this is that AODV drops packets in a situation where it has no route whereas DSR does package salvaging. DSR does a better job in masking the node mobility for TCP, however, DSR drops a larger number of packets due to poor ARP buffer strategy than AODV. The types of packets lost are DSR routing messages, TCP acknowledgements and TCP data packets. Each kind of loss contributes with about a third to the total number of packets lost in this way. The loss rate co-varies with the window size independent of routing protocol used. The rule is: the larger the window size the higher the loss probability.

8.6 Realistic Experiment

The purpose of this scenario is to test a more realistic traffic pattern in a MANET environment. The traffic pattern tested is from a simple World Wide Web (WWW) traffic model by H. Abrahamsson and B. Ahlgren [24]. The model approximates the TCP traffic generated by the Hyper Text Transfer Protocol (HTTP) [27] as a consequence of a typical user surfing the Web. A user surfing the Web is reading or browsing the content of a page. The pages

are viewed in a program called a browser. A page may contain references to other pages, so called links. When a user follows a link, by clicking on it, the page referenced by this link is downloaded to the browser. HTTP is the protocol used to distribute pages from the server to the where the pages are located to the browser clients. In HTTP, a client establishes one or more TCP connections to the server and issues a request for a page. The server responds by sending the requested page to the client via this or these TCP connection. The WWW traffic model consists of two stochastic variables. The first variable describes the amount of time in seconds after a page has been downloaded until a user clicks on a link causing another download. The second variable describes the size in bytes of a download.

The movement pattern in this experiment is based on the model used in the Random Way-point experiment above. In a 1500 by 300 rectangular area 30 nodes move in a random manner. The experiments include scenarios where the mean speed is set to 2 m/s and 10 m/s. The communication pattern simulates one HTTP server and five browser clients generating TCP traffic according the the model in [24]. Each client engages in a session consisting of a maximum of 15 downloads. The experiment is evaluated with AODV and DSR, however, only one TCP version was used. The experiments made so far has proven that there is no advantage using TCP/Sack in MANETs and therefore TCP/Reno was chosen. The main reason why TCP/Sack does not give any gain is that the main cause for retransmission is due to expired timers. The Sack option is useful when the the loss is advertised by the receiver for large windows. The scenarios were run with two window sizes, 4 kBytes and 16 kBytes. The reason for this is that the transfers are generally quite small and not expected to reach full window capacity. Also, a window larger than 16 kBytes is inefficient in earlier experiments. Unlike the previous experiments the segment size is not fixed throughout the lifetime of a scenario. Some downloads are smaller than the maximum segment size. When this happens the maximum segment size is adjusted and set to the size of the download. However, when the size of the download is greater than the maximum segment size the maximum is used. The maximum segment size chosen for all of the scenarios is 512 Bytes. For some transfers, by the limitation of having only one segments size, a little overhead is introduced when trying to fit a download in a sequence of packets. However, the overhead is never more than one packet.

8.6.1 Measured throughput

The throughput for each TCP transfer averaged over the ten scenarios when the mean speed is 2 m/s is shown in Figure 36. The experiment consists of

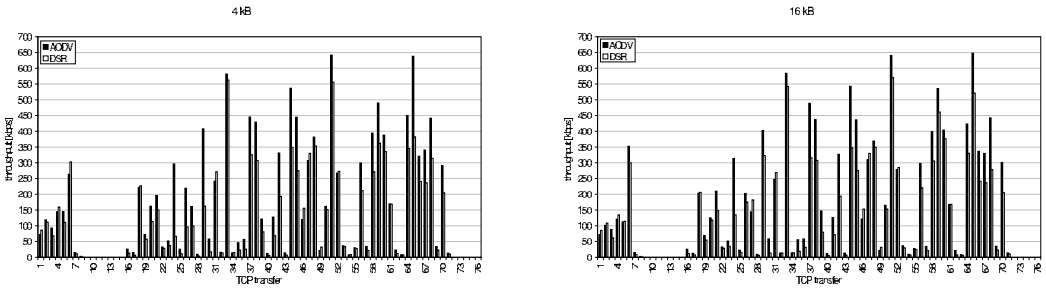

Figure 36: Throughput, mean speed 2 m/s.

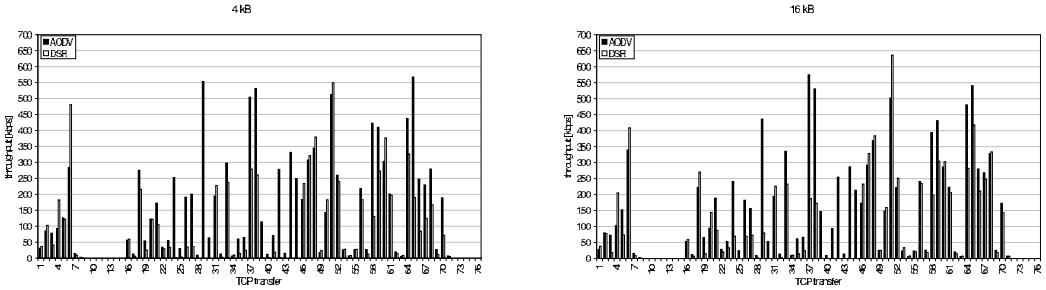

Figure 37: Throughput, mean speed 10 m/s

five clients acting in paralell and each client is responsible for 15 downloads. The first client's downloads are numbered from one to 15, the second client's downloads are numbered from 16 to 31 et cetera. The general impression is that the throughput varies a lot. The throughput ranges from zero to about 650 kbps, but most TCP transfers obtain a throughput less than 500 kbps. The reason why TCP transfers numbered 8 to 15 is zero is that the simulated user waits about 800 seconds until starting the ninth download. Since the maximum simulation time is 900 seconds these transfers are never started. A similar thing happens to TCP transfer numbered 72 to 75.

In almost all cases AODV performs better than DSR. AODV performs considerably better than DSR when the number of packets sent is larger. Transfer number 24, 29, 33, 37, 44, 51, and 65 all have in common that the number of segments sent is greater than 80 packets. In these cases, depending on the path length the byte overhead per packet is greater when using DSR and this slows performance. The difference in throughput when comparing the case when the maximum window is 4 kBytes and 16 kBytes is not great for either protocol. The transfers are generally small and never have use for a large window. However, when the transfers that are large there is a small increase.

Figure 37 shows the throughput obtained when the mean speed is 10 m/s. Compared to the 2 m/s case the throughput seems to be lower here.

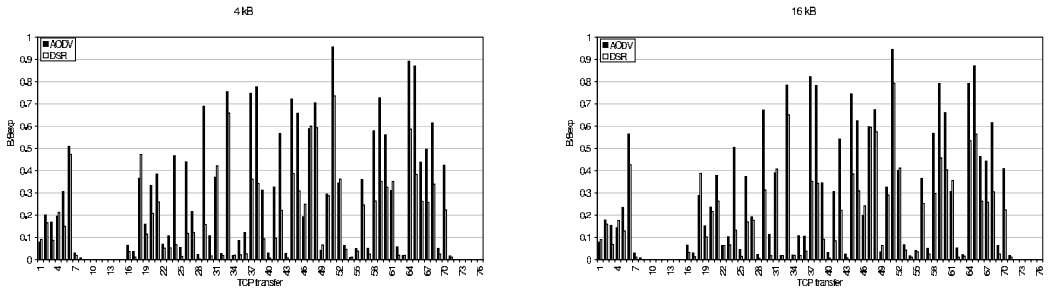

Figure 38: Expected throughput, mean speed 2 m/s.

The peaks in throughput obtained in the 10 m/s case is about 100 kbps higher than in the 2 m/s case. In earlier experiments there was an increase in throughput when the mean speed increased, however, the size of the transfers were considerable larger and was performed long enough to experience the benefits of shorter path lengths. Comparing the window size, there seems to be a small advantage using a larger window. The advantage in throughput is also greater when moving from 4 kBytes to 16 kBytes when the mean speed is 2 m/s.

Figure 38 shows the expected throughput obtained when the mean speed is 2 m/s. The values of expected throughput obtained when the window size is 4 kBytes range from about 0.95 to zero. However most TCP transfers experience a fraction of expected throughput below 0.5. In this experiment there is potentially five transfers performed at the same time. Depending on the movement pattern and the user download pattern, the contention for the medium may cause the throughput to drop if two or more transfers are synchronized. Generally, AODV obtains a better fraction of expected throughput than DSR does.The reason for this is that DSR has a greater packet byte overhead and has a poor ARP buffer strategy. Comparing the cases when the window size is 4 kBytes and 16 kBytes there is no conceivable difference in expected throughput for either protocol. Only a few larger transfers are able to utilize a greater window to boost throughput.

Figure 39 shows the expected throughput when the mean speed is 10 m/s. The expected throughput is lower than in the 2 m/s case. The peaks are lower here and there are more values below a fraction of 0.1. As in the 2 m/s case above AODV shows a better performance than DSR. The fraction of expected throughput is more or less the same when the window size is 16 kBytes compared to the case when the window size is 4 kBytes for both protocols. As above, only large transfers benefit from a large window size.

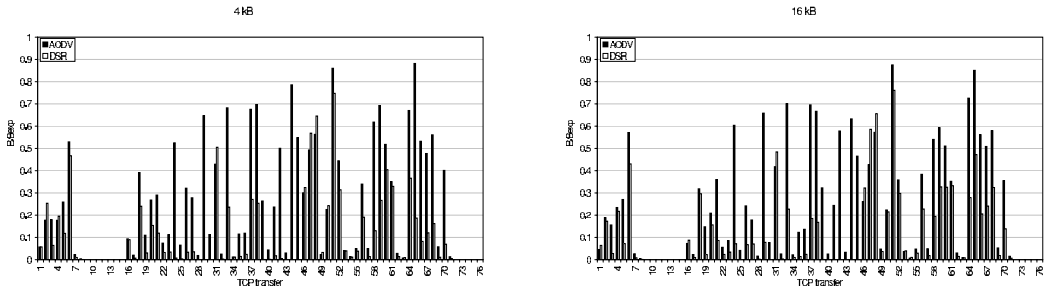

Figure 39: Expected throughput, mean speed 10 m/s.

8.7 Summary

The Realistic experiment has shown an example of that the MANET environment gives highly unpredictable throughput performance for WWW TCP traffic. The combined effect of node mobility and simultaneous downloads are the main causes for this.

Window size is not a big issue in networks where the main traffic is WWW TCP traffic. Most downloads are quite small, the median of the stochastic variable governing the size of the download is 7145 bytes [24], or about 14 packets at the size of 512 Bytes. A TCP transfer of this size can never reach full window capacity of 32 packets.

Mean node speed, on the other hand, seems to hurt throughput performance. The reason for this is that small downloads are vulnerable to node mobility. Since mobility is random and independent of download size mobility induced loss hurts small as well as large transfers at the same rate. Even though this rate is small, a few packets lost make out a substantial part of the total number of packets sent when the transfer size is small. A large fraction of packets lost cause poor performance.

In the Realistic experiment, a network routed with AODV gives better performance than a network routed with DSR. The reason for this is AODV's lower byte overhead and DSR's poor ARP buffer strategy. However, both protocols had higher throughput values for many transfers when the size of the transfers were larger than 80 packets.

9 Conclusion

TCP throughput performance in Ad Hoc networks is inversely proportional to the path length. For all paths shorter than six hops the throughput is halved when the path length doubles. This feature is unique for Ad Hoc networks but the following are not. The second biggest impact on throughput performance

is the presence of multiple transfers. The throughput is inversely proportional to the number of multiple parallel transfers. Two identical transfers sharing a common hop make the experienced throughput for both transfers drop to a half. Segment size has the third biggest impact on throughput performance. A large segment size gives a greater throughput. TCP and MAC layer protocol add byte overhead to each packet. The overhead is constant for each and independent of size. The extra overhead causes longer transmission times and gives lower throughput performance. A large amount of packets thus gives a larger amount of total overhead which in return lowers throughput performance.

Window size affects the loss rate but the loss rate is not strongly correlated to throughput performance. A large loss rate indicates a large amount of route changes in the network. The TCP flow control mechanism is sensitive to route changes. TCP may either time out due to lengthy routing procedures or time out due to packet loss. Sometimes these causes are combined. However, the previously mentioned indicators of throughput effect performance at an order of magnitude greater than the loss rate. The loss rate can be used to anticipate throughput as long as there is no mobility. The basic assumption that packet loss happens due to router overflow is not valid in a MANET. As soon mobility is introduced in the network, a large variety of package loss emerge and makes throughput estimation hard. Also, the TCP flow control mechanism itself works on the assumption that packet loss is due to router overflow. However, the amount of packets lost due to router overflow is only a small fraction of the total number of packets lost. The amount of packets dropped and drop causes are routing protocol dependent. AODV drops a lot of packets when it faces a situation where it has no route to the destination. DSR has been observed to drop a lot of packets in the ARP buffer.

Using TCP Selective acknowledgement gives no extra throughput performance. The Sack option gives the opportunity to retransmit more than one lost packet per round-trip without timing out. However, this has not showed to be of any help in MANET. The main reason that most retransmissions are due to that the retransmission timer expires. Also the TCP window is seldom big for long periods of time for the Sack option to make a difference.

Finally, there are some small differences in throughput performance when using different routing protocols. AODV has generally a larger loss rate than DSR but this performance loss is gained in the lower routing overhead. However, in a network where WWW traffic is present, AODV gives a little higher throughput. No protocol is able to mask node mobility effects to TCP.

A Installing the Simulator

Both the raw ns-2 simulator from Berkeley and the Monarch extension from CMU is under constant change. There are several research projects around the world that keep adding features and upgrading the specific details of the simulator. The implementations of the MANET routing protocols will change, since they are based on material marked as work-in-progress. Before installing the software, make sure you have a clear picture of what you need and then download all those bits and pieces that are most up to date. A good starting point is probably to check out the IETF MANET [1] charter home page or the Monarch project home page [18].

This installation guide has been tested as late as March 2000 on a Red Hat Linux system and the aim for it is to end up with the same environment I used for my experiments. The installation guide is for a simulator based on ns-2 version 2.1b6, Monarch's extension to ns-2 version 1.1.2 and AODV supporting the fourth IETF Internet draft. There is no guarantee that it will work on any other platform or later versions of the code listed above.

A.1 Downloading

Here follows a guide to where the required software can be found. Download all pieces of software before you begin installing them.

1. Download the ns-2 simulator. Download the all-in-one bundle. It can be found at the ns home page at URL:
   `http://www-mash.cs.Berkley.EDU/ns/ns-build.html#allinone`

2. Download the Monarch's extension to ns-2. You can find it on the following URL:
   `http://www.monarch.cs.cmu.edu/cmu-ns.html`

3. Download AODV code. At the time of writing, it can be found on URL:
   `http://www.ececs.uc.edu/~mmarina/aodv/`

4. Download the tar archive containing bug fixes, trace extension etc. from this project's home page:
   `http://www.sics.se/~mattiaso/`

A.2 Installing

These are the steps you should take in order to end up with the same simulator environment used in this study. Make sure you take steps in the order suggested.

1. Unszip tar archive containing the ns-2 simulator and extract files from it . Use `gzip -d ...` to unpack and `tar xvf ...` to extract.

2. Descend the `ns-allinone-2.1b6` directory and type `./install`.

3. When the configuring and compiling finishes, add the environment variable `LD_LIBRARY_PATH` with the path to the `/otcl_1.0a5/` directory.

4. Unpack and extract the files from the tar archive with the Monarch's extension to ns-2 within the `ns-allinone-2.1b6` directory. Descend the `ns-src` directory.

5. Unpack and extract the files from the archive with the AODV code in the `ns-src` directory.

6. Unpack and extract the files in the bug fix archive.

7. Open the file `configure` with your favorite editor.

8. At row 2208, change `OTCL_VERS=1.0a4` to `OTCL_VERS=1.0a5`.

9. At row 2381, change `TclCl_VERS=1.0b8` to `TclCl_VERS=1.0b9`. Save the file and exit the editor.

10. At the shell prompt. Type `sh configure` and hit enter.

11. Open and edit the file `trace.cc`. Change line 330 where it reads `var->value(tmp)` to `var->value(tmp,256)`.

12. Open and edit the file `agent.cc`. Change line 150 and line 221 where its says `var->value(tmp)` to `var->value(tmp,128)`.

13. Open file `adhockey-slaver.cc` and comment out line 53.

14. Open and edit the file `Makefile`.

15. At line `-ldl` to line 56.

16. At line 70 change `TCLSH=/tcl8.0/unix/tclsh`, to `TCLSH=/tcl8.0.4/unix/tclsh`.

17. Before line 173 insert the following line:
`cmu/aodv/aodv_rtable.o cmu/aodv/aodv_rqueue.o \\`

18. At line 180 insert line `cmu/zero/zero.o`. Save the file and exit the editor.

19. Type `make depend; make` and hit enter.

20. Enter the `ad-hockey` directory in the `ns-allinone-2.1b6 directory`. Type `make`.

21. Open the file `viz-trace`. At line 27 replace
    `if(/^[srf]/){` with
    `if(/^[srf]/ && !/MAC/ ) {`.
    You're finished installing the code.

B Simulator Trace File Format

At the MAC layer, router layer and agent layer, certain events are being monitored by default. These events are summarized and outputed to a log line. The log lines are collected in the trace file and is the ultimate output of the simulator. The trace file is divided in up to three parts. First we find a three line header confirming scenario and communication pattern files etc. used in the trace, then there may be a section reporting routing protocol options. Finally, the event log lines of the trace follows.

B.1 Log File Lines

The event log lines found encountered in this study consists of several logical parts. The contents of these parts depend on what event generated the row. To distinguish events, each row is opened with a unique single character. The value of this character may depend on the routing protocol used, however, the most common values for this character are `s`, `r`, `f`, `D`, and `M`. These characters are abbreviations for the following events: `s` - packet is sent; `r` - packet is received; `f` - packet is forwarded; `D` - packet is dropped; and `M` - movement of nodes. If a row opens with either `s`, `r`, or `D`, this character will be followed by at least eight parts describing the some basic information of the event. Depending on the event there may be more parts after the initial eight parts.

Any log line that opens with the character `S` contains information of the internal state of DSR. The opening character is followed by a short string giving hints of what is reported. This string is followed by a time stamp and the node number where the event occurred, then event-dependent information is found.

B.2 Basic Log Line

The eight initial parts are:
(time of event) (node identification number) level (drop reason) (unique packet identification number) type (packet length) (MAC layer information).
The meaning of each field is:

- time of event - a floating point value of the time the event was generated.
- node identification number - an integer enclosed in underscore characters (i.e. _13_) identifying the node at which the event occurred.
- level - a string consisting of three upper-case characters describing at what level in the network stack the event occurred. `AGT` stands for agent level; `RTR` is router level and `MAC` is short for Medium ACcess layer.

- drop reason - if this line begins with the letter `D` this field consists of string of three uppercase letters describing the cause of this packet being dropped. See table 11 for a detailed description of drop reasons. The values of this string depends on the level at it was generated. If this line opens with either `s` or `r`, this field consists of three dashes (i.e. `---`).

- unique packet identification number - each level in the network stack associates a unique identification number to each packet being handled.

- type - a string describing the type of this packet. In this study the following types will be encountered: `tcp`, `ack`, `DSR`, `ARP`, and `MAC`. `tcp` indicates that this packet is a TCP data segment; `ack` - this is a TCP acknowledgement; `DSR` states that this is a DSR message; `AODV` indicates that this is an AODV message; `ARP` says its either a ARP request or reply; and finally, `MAC` tells that it is a MAC layer message.

- packet length - this is the length of the packet in bytes.

- MAC layer information- This part is further divided into five separate fields with data taken from the 802.11 frame header. The data displayed is in the following format: `[` (frame control) (duration) (destination address) (source address) (body) `]` meaning:

  - frame control - Is an integer in hexadecimal notation. The four first low end bits indicate what message this line logs. The four bits corresponds to last digit of this integer. A value of `B` means RTS - request to send , `C` is CTS - clear to send, `D` is ACK - acknowledgement and `0` says is a DATA frame.
  - destination address - MAC address of the receiver of this frame.
  - source address - MAC address of the sender of this frame.

B.3 Agent Level Events

If the level field contains the value `AGT` the initial eight fields are followed by four parts. The first is a sequence of seven dashes (`-------`). This part is followed by information taken from the IP header in the following format: `[` (source address) : (source port) (destination address) : (destination port) (time to live) (next hop) `]`. This fields can be understood as follows:

- source address - the address of the host where this packet was originated.

Code	Meaning
ARP	Packet dropped by ARP.
COL	Collision in MAC layer transfer.
END	Packet dropped since simulation ends.
IFQ	Interface queue is full.
NRTE	Packet dropped since there was no route for it.
RET	Packet dropped since MAC retransmission count was exceeded.
TOUT	Packet expired.
TTL	Value in TTL field (in IP header) reached zero.
CBK	MAC layer call back on a bad link.

Table 3: Common drop reasons

- source port - identifying process on the source host .
- destination address - the destination host address of this packet.
- destination port - identifying process waiting for this packet.
- time to live - the remaining number of hops this packet is allowed to make.
- next hop - the address to the host next in the route.

B.4 TCP Events

If the level field is set to either `tcp` or `ack` the IP specific information is followed by information extracted from the TCP header. This information is presented as follows: `[` (sequence number) (acknowledge number) `]`. These fields explained:

- sequence number - the sequence number of this TCP segment
- acknowledge number - the acknowledge number of this TCP segment

Finally, the fourth and last part consist of two fields supplied by the simulator. They are (number of forwards) and (optimal number of forwards). The number of forwards field contain the number of times this packet was forwarded and the optimal number of forwards is the optimal number forwards this packet needed in order to reach its destination.

B.5 Router Level Events

If the level field contains the value RTR the initial eight fields are followed by at least two parts. The first and second part consists of a sequence of dashes and IP specific information in the same format as described in 7.3 above. This information is followed by data specific for each protocol.

B.6 DSR Packets

If the type field is set to DSR we find four parts immediately following the initial router level information. This information is extracted from the DSR message header. The first part is a single integer stating the number of addresses in the route for this package.

The second part is further divided into three fields. They are: [(route request) (route request sequence number) (max propagation)] and can be understood by:

- route request - does this message contain a route request?
- route request sequence number - if this is a route request it's sequence number is given, otherwise this field is set to zero.
- max propagation - the number of time the route request message is allowed to be forwarded.

The third part is set only if the message contain a route reply. It has the following format: [(route reply) (route request number) (route reply length) (source) – (destination)].

- route reply - does this message contain a route reply?
- route request number - the reply is due to this request.
- route reply length - the number of nodes in the route proposed in this message.
- source - the first address in this route.
- destination - the last address in this route.

The forth and final part is set if the message includes a route error. It is formated as follows: [(route error) (number of route errors) (tell host address) (from host address) – (to host address)]. Explanation:

- route error - does this message carry a route error notification?
- number of route errors - the number of route error in this message.
- tell host address - tell this host that the host specified by from host address no longer can forward packets with destination to host address.
- from host address - host no longer capable of forwarding packets
- to host address - destination that no longer can be reached on a particular route.

B.7 AODV Packets

If the type is `AODV` there are two parts trailing the general router level information. The first part contains information taken from the AODV header and the second part indicates what header is displayed. There are four valid values for this part: `(RREQ)` - route request message; `(RREPL)` - route reply message; `(RERR)` - route error message; and `(HELLO)` - which indicates a hello message.[1]

If the second part is set to `(RREQ)`, the first part has the following format: `[` (request type) (hop count) (broadcast id) `[` (destination address) (destination sequence number) `]` `[` (source address) (source sequence number) `]` `]`.

- request type - a number in hexadecimal form indicating the type of the route request.
- hop count - The number of hops from the source to the node currently handling the request.
- broadcast id - A sequence number which together with the IP address uniquely identities this route request message.
- destination address - IP address of destination
- destination sequence number - The last sequence number received in the past for any route to the destination.
- source address - IP address of source.
- source sequence number - Current sequence number to be used for route entries pointing to the source.

[1] The log output format is specific to my changes of `cmu-trace.cc`.

If the value is either (RREPL) or (HELLO) the first part is formated as follows: [(reply type) (hop count) [(destination) (destination sequence number)] lifetime]. Each field can be understood by:

- reply type - a number in hexadecimal indicating the type of route reply.
- hop count - the number of hops from the source to the destination.
- destination address - the IP address of the destination for which a route is supplied.
- destination sequence number - the destination sequence number associated to this route.
- lifetime - the amount of time this route should be considered valid.

If the second part is set to (RERR), the first part has the following format: [(error type) (hop length) (unreachable destination)].

- error type - a number in hexadecimal indicating the type of route request.
- hop length - the number of destinations which are unreachable
- unreachable destination - the first unreachable destination in the list of unreachable destinations.

B.8 Other Router Level Events

If level is set to either tcp or ack the rest of the line contains information taken from the TCP header as described in 7.4 above. These events signals that a TCP data or acknowledgement segment was received by or sent to the agent level or the link layer from the router layer.

B.9 MAC Layer Events

If the level is set to MAC the event happened at the MAC layer. It is possible to see what kind of data is carried in a MAC layer data frame by looking at the type field. If the type is set to tcp, ack or DSR, the information that follows the eight initial fields are on the form described above. If the type is set to ARP we find, after seven of dashes, data taken from the ARP header.

B.10 ARP Events

The ARP events are logged as follows: [(message type) (source MAC address)/(source IP address) (destination MAC address)/(destination IP address)]. Each field explained:

- message type - Is either `REQUEST` or `REPLY`
- source MAC address - Sending nodes MAC layer address.
- source IP address - This is the sending nodes IP address.
- destination MAC address - If message type is set to `REQUEST` this field is set to zero. Otherwise it contains the MAC address of destination.
- destination IP address - Here, the IP address of the node ARP wants to learn the MAC address, is found.

References

[1] Mobile Ad-hoc Networks (MANET) charter. http://www.ietf.org/html.charters/manet-charter.html

[2] S. Corson and J. Macker, Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations, draft-ietf-amnet-issues-*.txt, Work-in-progress.

[3] J. Schiller, Mobile Communications, Addison-Wesley, Great Britain, 2000.

[4] Andrew S. Tannenbaum, Computer Networks, Prentice-Hall, New Jersey, 1996.

[5] IEEE. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (IEEE 802.11), IEEE, 1997

[6] Elizabeth M. Royer and Chai-Keong Toh, A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks, IEEE Personal Communications, April 1999.

[7] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. Proceedings of Mobicom'98, Dallas.

[8] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek and Mikael Degermark, Scenario-based Performance Ananlysis of Routing Protocols for Mobile Ad-hoc Networks. Proceedings of Mobicom'99, Seattle.

[9] Samir R. Das, Charles E. Perkins and Elizabeth M. Royer. Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. Proceedings of Infocom 2000, Tel-Aviv.

[10] David A. Maltz, Josch Broch, David B. Johnson, Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed, CMU School of Computer Science Technical Report CMU-CS-99-116. March 1999.

[11] Laura M. Feeney. A Taxonomy for Routing Protocols in Mobile Ad Hoc Networks, SICS Technical Report T99/07, October, 1999.

[12] Josh Broch, David B. Johnson and David A. Maltz, The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks, draft-ietf-manet-dsr-03.txt, Work-in-progress.

[13] Charles E. Perkins, Elizabeth M. Royer and Samir Das. Ad Hoc On Demand Distance Vector (AODV) Routing, draft-ietf-manet-aodv-04.txt, Work-in-progress.

[14] W. Richard Stevens, TCP/IP Illustrated Volume 1: The protocols, Addison-Wesley, Reading, Massachusetts, 1996.

[15] M. Allman, V. Paxson and W. Stevens, TCP Congestion Control, April 1999, RFC 2581.

[16] M. Mathis, J. Mahdvi, S. Floyd and A. Romanov, TCP Selective Acknowledgement Options, October 1996, RFC 2018.

[17] K. Fall and K. Varadhan, ns Notes and Documentation, The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC, http://www-mash.CS.Berkeley.EDU/ns/, Work-in-progress.

[18] The CMU Monarch project, The CMU Monarch Project's Wireless and Mobility Extension to ns. http://www.monarch.cs.cmu.edu/, Work-in-progress.

[19] M. Allman and A. Falk, On the Effective Evaluation of TCP, ACM Communication Review, October 1999.

[20] J. Padhye, V. Firoiu, D. Towsley and J. Kursoe. Modeling TCP Throughput: A Simple Model and its Empirical Validation. Proceedings of SIGCOMM'98, Vancouver, British Columbia, Canada, 1998.

[21] G. Holland and N. Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks, Proceedings of Mobicom'99, Seattle.

[22] M. Gerla, K. Tang and R. Bagrodia, TCP Performance in Wireless Multi-hop Networks, Proceedings of IEEE WMSCA'99, New Orleans.

[23] H. Abrahamsson, Traffic Measurement and Analysis, SICS Technical Report T99/05. September 1999.

[24] H. Abrahamson and B. Ahlgren, Using Empirical Distributions to Characterize Web Client Traffic to Generate Synthetic Traffic, To appear in Proceedings of GlobCom, San Francisco, 2000.

[25] Jean-Paul M.G. Linnartz, Wireless Communication CD-ROM Home Page, http://wireless.per.nl/multimed/cdrom97/contents.htm, April 2000.

[26] Phil Karn, MACA - A New Channel Access Method for Packet Radio. Proceedings of the 9th ARRL Computer Networking Conference, London, Ontario, Canada, 1990.

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, June 1999, RFC 2616.